行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※ 可容錯之行動物件定位演算法 ※
※ Fault-Tolerant Tracking of Mobile Objects ※
※ ※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：☑個別型計畫　　□整合型計畫
計畫編號：NSC 90－2213－E－002－104
執行期間：90 年 8 月 1 日至 91 年 7 月 31 日
計畫主持人：蔡益坤

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

執行單位：國立臺灣大學資訊管理學系

中　華　民　國　92　年　4　月　22　日

行政院國家科學委員會專題研究計畫成果報告

可容錯之行動物件定位演算法

Fault-Tolerant Tracking of Mobile Objects

計畫編號：NSC 90-2213-E-002-104
執行期間：90 年 8 月 1 日至 91 年 7 月 31 日
主持人：蔡益坤 (Yih-Kuen Tsay) 國立台灣大學資訊管理學系

摘要

找出特定物件位置以便遞送欲傳給該物件之訊息，是所有分散式系統都應有能力完成的工作。多數支援行動物件的系統利用所謂的 home-based 策略來解決定位問題。在這類方法中，所有的物件都有一個固定的家，欲傳給某一物件的訊息必須先取道該物件的家。這類方法有容錯上的弱點，當一個物件的家損壞了或是到這個家的所有通訊路線都損壞了，那系統中的使用者便無法再操作該物件。

我們的主要研究目的便在探討容錯能力好的追蹤物件位置的分散式目錄結構及定位演算法。這項研究是我們稍早的一個名為「分散式互斥與物件追蹤」的國科會研究計畫的延續。在該計畫裡，我們探討如何將分散式互斥演算法應用到行動物件的定位工作。我們從既有互斥演算法推導出兩個定位演算法；我們的主要貢獻在於處理容錯性的問題。

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、複製物件、路由協定。

Abstract

Locating an object so as to deliver messages intended for the object is a basic task of every distributed system. Most systems that support mobile objects use home-based tracking strategies wherein every object has a designated home and all messages for an object (or at least the first one in the same session) are routed through its home. One problem of this type of protocols is that an object becomes inaccessible when its home fails or communications to its home are blocked.

We investigate distributed directory structures and algorithms for tracking mobile objects that exhibit a high degree of tolerance to node and communication failures. This research is a continuation of a previous NSC-sponsored project titled ``Distributed Mutual Exclusion and Object Tracking" (NSC 89-2213-E-002-021) where we explored the idea of adapting token-based mutual exclusion algorithms for object tracking. We derive from existing token-based algorithms two schemes for tracking mobile objects. Our main contribution lies in addressing fault tolerance issues.

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

Fault-Tolerant Tracking of Mobile Objects (Final Report of NSC 90-2213-E-002-104)

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

Abstract

Locating an object so as to deliver messages intended for the object is a basic task of every distributed system. Most systems that support mobile objects use home-based tracking strategies wherein every object has a designated home and all messages for an object (or at least the first one in the same session) are routed through its home. One problem of this type of protocols is that an object becomes inaccessible when its home fails or communications to its home are blocked.

We investigate distributed directory structures and algorithms for tracking mobile objects that exhibit a high degree of tolerance to node and communication failures. This research is a continuation of a previous NSC-sponsored project titled "Distributed Mutual Exclusion and Object Tracking" (NSC 89-2213-E-002-021) where we explored the idea of adapting token-based mutual exclusion algorithms for object tracking. We derive from existing token-based algorithms two schemes for tracking mobile objects. Our main contribution lies in addressing fault tolerance issues.

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

1 Introduction

One primary function of a distributed system is to store and manage certain resources so that they can be conveniently shared by users of the system. We use the generic term *object* to refer to an instance of any of these shared resources. Locating an object so as to deliver messages such as operation requests intended for the object is therefore a fundamental task of most distributed systems. Efficient performance of this object tracking (locating) task is nontrivial as nodes and communication links in a distributed system may fail. It becomes even more complicated when objects may move. We briefly outline an abstract model to provide the problem setting and explain the issues of main concern in tracking the location of a mobile object.

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.

Failures may occur in such a network. When a node is failing, we assume that it will send a special notice message to each of its neighbors and then stop; when a node recovers, it also sends out notices and then starts operating from a predefined initial state. Similarly, when a link is failing, it will send a special notice to each of its two end-nodes and then stop (all other messages in transit are lost); when a link recovers, it also sends out notices and then starts operating with no other messages in the link. In other words, we are adopting the detectable fail-stop failure model with possible recovery.

The task of locating an object in a distributed system is usually performed by a directory service or name service of the system [CDK01]. The relevant directory service may be centralized at a particular server or distributed across a number of servers or even the entire system. Typically, certain distributed data structure has to be collectively maintained by nodes of the system that serve as the directory for keeping track of the objects.

Most existing systems that support mobile objects use a home-based tracking scheme wherein 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. An object becomes inaccessible when its home fails or communications to its home are blocked. The home is a potential bottleneck of performance.

Our primary goal is to find distributed directory structures and algorithms for tracking mobile objects that exhibit a high degree of tolerance to node and communication failures. This research is a continuation of a previous NSC-sponsored project titled "Distributed Mutual Exclusion and Object Tracking" (NSC 89-2213-E-002-021) where we explored the idea of adapting token-based mutual exclusion algorithms for object tracking [Hua00]. We derive from existing token-based algorithms two schemes for tracking mobile objects. Our main contribution lies in addressing fault tolerance issues.

Related Work

There has been a considerable amount of work on object tracking. (we use the term object tracking in this research to emphasize that the object being located may move). We consider only algorithms that maintain a *dynamic* distributed directory structure (in contrast with home-

based algorithms). We include some token-based mutual exclusion algorithms, as they can be adapted for tracking mobile objects (the token moves very much like a mobile object does). A summary of the more noticeable work is given below. Some of these works make use of an underlying routing service, while some maintain all necessary routing information. Most of them use tree-like directory structures, which cannot sustain failures of communication links that are tree edges. Kutten and Porat [KP99] presented an algorithm for maintaining a spanning tree in a network with possible link failures. It may be possible to incorporate this type of algorithms in tree-based object tracking algorithms to provide the needed tolerance to link failures.

Van de Snepscheut [VdS87] was probably the first to give solutions to the mutual exclusion problem in a network of processes, extending the earlier work for rings by Martin [Mar85]. The main idea was to orient the edges of the network so that they point to the token holder; when the token moves, the directions of the edges are updated accordingly. Raymond [Ray89] proposed another algorithm which turned out to be identical to a restricted version of Van de Snepscheut's where a rooted spanning tree instead of a directed acyclic graph is maintained. However, he was able to give a more detailed analysis of the average case message complexity, which is $O(\log N)$.

Naimi *et al.* [NTA96] presented yet another spanning tree-based solution for networks whose topology is a complete graph. Unlike the previous algorithms, the edge set of the tree maintained by their algorithm changes over time. They 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. They noted the close relationship between distributed mutual exclusion and object tracking. Their main algorithmic technique can be seen as a combination of those of Van de Snepscheut and Naimi *et al.*

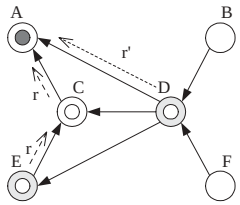
An early work by Mullender and Vitányi [MV88] had suggested the relationship between distributed mutual exclusion and object tracking (or name service, in their terminology). They showed that mutual exclusion and name service can be formulated as special instances of the so-called distributed match-making problem. However, the formulations seem to be biased toward particular types of algorithms.

PRR (Plaxton, Rajaraman, and Richa) [PRR97], Chord [SMK+01], CAN [RFHK01], Tapestry [ZKJ01], and Simplified PRR [LP02] represent a most recent line of research that focuses on achieving scalability. These location schemes either do not handle mobility of an object or simply treat it as a combination of object deletion and insertion. Moreover, these schemes can all be classified as home-based.

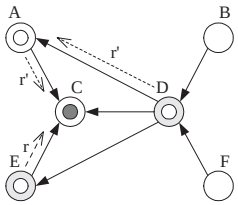
2 Scheme 1

The basic construction of Scheme 1 is derived from Van de Snepscheut's algorithm for mutual exclusion on a network. Our contribution lies in the handling of node and link failures.

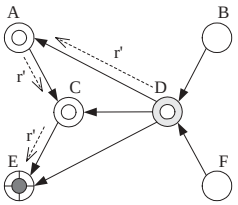
The edges (or a suitable subset of the edges) of the network are oriented to form a single-sink *directed acyclic graph* (DAG) with the object holder as the sink. (Note that the DAG may be chosen to cover a suitable subset of the edges if there is a concern of maintenance cost; but, we assume that all nodes participate in sharing the object and should be covered by the DAG.) When a node wishes to acquire the object, it sends a request along one of the directed paths to the object holder. The next node in the path will either relay the request along one of its outgoing edges or put the request in a local queue. The object is routed in the reverse order along the path that the request has travelled. The direction of each edge incident to a node is updated to point to the node, when it receives the object; this maintains acyclicity and also ensures that the object holder is the only sink of the entire graph. Conflicting requests may occur. Each node (as an originator or forwarder) permits at most one outstanding request while keeping all others in its local queue. A scenario is shown in Figure 1.



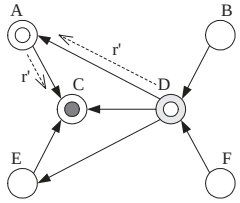
(a) Request r is initiated by E and r' by D. Request r arrives at A earlier.



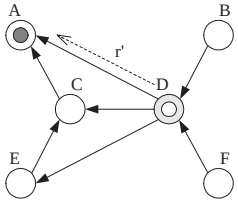
(b) A sends the object to C and also relays request r' to C. (C, A) is redirected towards C.



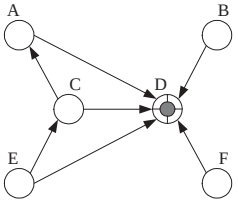
(c) C relays the object and request r' to E. (E, C) is redirected towards E.



(d) E releases the object to C. (C, E) is redirected towards C.



(e) C relays the object to A. (A, C) is redirected towards A.



(f) A relays the object to D. (D, A), (D, C), and (D, E) are redirected towards D.

Figure 1: How Tracking Scheme 1 works.

We now consider node failures (according to the fail-stop model). If a node has sent a request to one of its neighbors that fails, it sends the request again to a second neighbor if there is

any; otherwise, it waits until the failed neighbor recovers. The DAG maintained by the tracking scheme provides alternative directed paths for a node to the object holder. (All directed paths from a node to the object holder may be cut off by the failure of a node that happens to lie at a intersection of all the paths.) When a failed node recovers, it inquires its neighbors to restore the directions of all incident edges. Assuming a stable storage for backing up the object, a failed object holder may recreate the object when it recovers and finds itself to be the object holder.

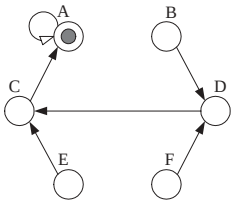
We next consider link failures. If a node has sent a request through a link that fails, it sends the request again through a second link directed towards the object holder if there is any; otherwise, it waits until the failed link recovers. The harder problem is for the two processes at the ends of a failed link to decide its orientation when the link recovers. During the period of time from the failure to the recovery of a link, its two end-processes may go through a number of state changes, in particular, holding and releasing the object. The orientation of a recovered link may be determined according to the numbers of times that the two end-processes have held the object since they detected the last failure of the link.

3 Scheme 2

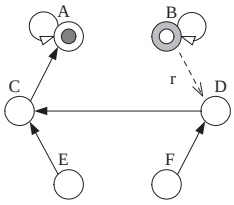
The basic construction of Scheme 2 is derived from the algorithm of Naimi *et al.* for mutual exclusion on a network. Again, our contribution lies in addressing fault-tolerance issues. In this scheme, we assume an underlying routing service that provides a reliable end-to-end communication (like the Internet Protocol) for the network of processes. The purpose for this is to relegate the fault-tolerance responsibility regarding message communications to the routing service. The network of processes *that participate in sharing the mobile object* now turns into a logically complete graph (a node may send a message "directly" to any other node via the routing service).

Scheme 2 maintains a dynamic tree structure for routing requests, whose set of tree edges (which are logical links) changes over time. It also maintains an additional distributed queue that tells the current object holder and subsequent holders which node is the next in line waiting for the object (in contrast, Scheme 1 uses separate local queues). Conceptually, the root of the routing tree is also the tail (and sometimes the head as well) of the distributed queue, where some other process may be added (what really happens is more complicated with possibly multiple trees and roots and hence multiple queues being created and merged).

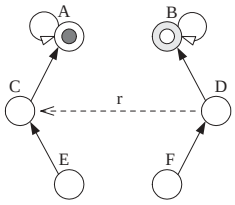
Initially, a rooted spanning tree of the network is formed with the object holder as the root. The initial object holder is also the only node, the head, and the tail of the distributed queue. When a node wants to acquire the object, it sends a request to its parent. It then regards itself as the new root, expecting that, once its request is received by the current root, it will be added to the distributed queue and become the new tail of the queue. Other nodes on the directed path to the current root, upon receiving the request, make the request originator their



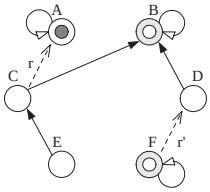
(a) A holds the object and is the root, treating itself as its parent.



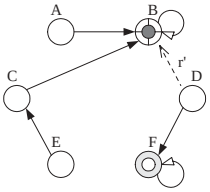
(b) B sends a request r to its parent D and makes itself a new root.



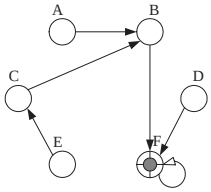
(c) D relays request r to its parent C and makes B its new parent.



(d) C relays request r to its parent A and makes B its new parent. A will add B to the queue when it receives r. F also initiates a request r'.



(e) A releases the object to B (via the assumed routing service) and makes B its new parent. D forwards request r' to B.



(f) B releases the object to F, and makes F its new parent.

Figure 2: How Tracking Scheme 2 works.

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. Finally, the current root inserts the request originator behind itself in the queue and also makes the originator 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 object for a finite amount of time, will pass out the object to the next node and remove itself from the queue. Every node in the queue eventually will get the object. A scenario can be found in Figure 2.

As shown in the figure, more than one processes may be trying to acquire the object. Though the overall changes to the tree and the queue may be more complicated, processes behave just as described in the preceding paragraph. 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 back to the distributed queue when the new root is eventually added to the queue. A root 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 request originator; a second request will be forwarded to the new parent.

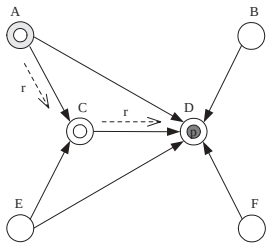
With the underlying routing service, a (logical) link failure occurs when the service fails to deliver a message from one end to the other of the link. Assuming that the routing service

makes use of any possible path, the failure implies that there is currently no effective path in the *entire* network connecting the two end-processes. If a requesting process is one of the two ends and its parent is the other, then no effective directed path from the requesting process to the object holder exists in the dynamic tree. It can be concluded that the process simply has to wait until the routing service succeeds in delivering the request message.

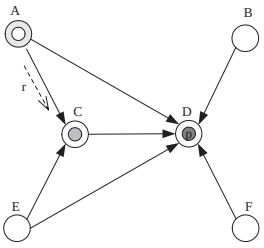
The scheme is vulnerable to the failure of a node that participates in sharing the mobile object.

4 Tracking Replicated Objects

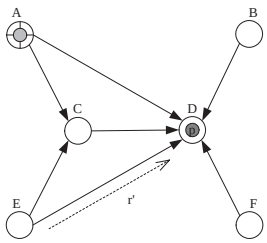
Objects may be replicated for higher availability (fault-tolerance) or better performance. We consider primary-secondary (primary-backup) replication of objects. One copy of the object is designated as the *primary* copy. A process wishing to write the object must first acquire the primary copy and then "invalidate" the *secondary* copies (without actually moving them). Read operations can be performed with any secondary copy. We extend Scheme 1 to track such replicated objects.



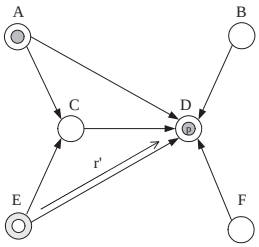
(a) D holds the primary copy. A initiates a request r to C, which relays it to D.



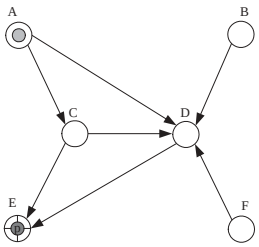
(b) D creates a secondary copy and sends it to C.



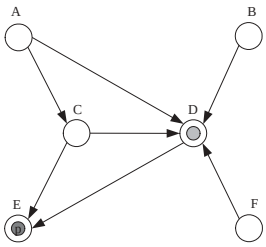
(c) C relays the secondary copy to A. E initiates a request r' to D.



(d) D sends the primary copy to E. (E, D) and (E, C) are redirected towards E.



(e) A returns the secondary copy to D (A may also send it via C).



(f) D relays the secondary copy to E.

Figure 3: Tracking primary and secondary replicas.

Initially, the *primary* copy is assigned to one of the processes in the network. A counter is kept

in the primary copy to record the number of *secondary* copies currently in the system so as to guarantee that there are at most $k-1$ of them at a time. The tracking scheme maintains a single-sink directed acyclic graph of the network such that all edges are directed toward the primary copy holder. Any process holding either the primary copy or a secondary copy may enter the critical section. A request is routed along the directed edges until it reaches the primary copy holder; any path may be chosen if there exist more than one paths to the primary copy. If the primary copy is not in use, it is sent to the request originator; the directions of the edges change accordingly as the copy moves. Otherwise, the primary copy holder generates a secondary copy for the request originator or, if the counter has reached $k-1$, put the request in a local queue. The movement of a secondary copy does not change the direction of an edge. The basic ideas are illustrated in Figure 3.

5 Concluding Remarks

We have focused on solutions for tracking mobile objects that do not assign a fixed home (or multiple fixed homes, for a better degree of fault-tolerance) to an object. These "homeless" tracking schemes, unlike the home-based ones, seem to avoid the performance bottleneck of a home. However, if an object and its replicas require a separate directory structure, the schemes will not scale to a large number of different objects. The merits of such homeless schemes remain to be further studied.

Acknowledgment

This is a joint work with Mr. Chien-Chung Huang and Mr. Po-An Chen.

References

[CDK01] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, 2001.

[DH98] M.J. Demmer and M.P. Herlihy. The arrow distributed directory protocol. In *DISC '98, LNCS 1499*, pages 119–134, 1998.

[Hua00] C.-C. Huang. Exclusion algorithms and their applications in networks of processes. Master's thesis, Department of Information Management, National Taiwan University, June 2000.

[KP99] S. Kutten and A. Porat. Maintenance of a spanning tree in dynamic networks. In P. Jayanti, editor, *Distributed Computing: 13th International Symposium, DISC '99, LNCS 1693*, pages 342–355. Springer, September 1999.

[LP02] Xiaozhou Li and C.G. Plaxton. On name resolution in peer-to-peer networks. In *ACM POMC '02*, pages 82–89, 2002.

[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.

[PRR97] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM SPAA '97*, 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.

[RFHK01] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. In *ACM SIGCOMM '01*, pages 161–172, 2001.

[SMK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM '01*, pages 149–160, 2001.

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

[ZKJ01] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Computer Science Division, April 2001.