On Efficient Departure for Dynamic Asynchronous Systems

Sathya Peri Neeraj Mittal
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75083
sathya.p@student.utdallas.edu neerajm@utdallas.edu

Contact author: Sathya Peri
Department of Computer Science
The University of Texas at Dallas
Email: sathya.p@student.utdallas.edu

1 Introduction

Topology maintenance is an important problem in dynamic systems systems where processes join and depart particularly in structured peer-to-peer systems. Most of the work that has been done in the area of peer-to-peer computing impose and maintain a structure. The structures have varied from rings [9] to hyperdelta networks [5]. Li et al's paper [4] observes that there are two ways of topology maintenance: *passive* and *active*. In the passive approach, when the system topology changes, neighborhood relationships are not immediately updated. A repair protocol periodically executes in the background which restores the topology. In the active approach the neighbor pointers are immediately updated whenever a change occurs. One can see that in the passive approach if the topology changes are very quick, then there can be possibly several instances of time where the system can get disconnected. In the active approach where the neighborhood relationship is immediately updated as the topology changes, system disconnection can be circumvented.

In dynamic systems there are several applications of interest that demand that the system always stay connected or *perpetual connectivity* (provided by the active approach) as opposed to *eventual connectivity* (guaranteed by the passive approach). Some examples are stable property detection [3], [7], atomic and reliable broadcasts [1]. If the system gets disconnected, then these applications may no longer work correctly. In the rest of this document, we concentrate on perpetual connectivity provided by the active approach.

In this paper we study the problem of *efficient depart algorithm* for maintaining perpetual connectivity. In Section 2 we define the problem. In Section 3 we briefly explain the the lower bound on the depart algorithm.

2 Problem Description

In dynamic systems where processes join and depart, process joins do not affect system connectivity. It is process departures that can possibly disrupt the connectivity of the system. Whenever a process wishes to depart it has two options: i) it can depart quietly without informing anybody ii) it can depart gracefully after informing its neighbors. It is easy to see that case i) can not ensure perpetual connectivity. Hence only case ii) can be employed for maintaining perpetual connectivity. A process wishing to depart informs all its neighbors and then departs, the approach taken in [1], [3], [4], [7].

In the above approach a process wishing to depart updates its neighbors such that they become neighbors after its departure. For instance, consider processes p, q, r such that q and r are neighbors of p (but q and r are not neighbors of each other) and process p wishes to depart. Before departing p ensures that q and r become neighbors. In this way the system stays connected when a process departs. But when two

neighboring process wish to depart simultaneously then in order to maintain perpetual connectivity, their depart sequences must be serialized. For instance if processes p and q wish to depart at the same time, then p has to wait for q to depart and then depart (or the vice-versa). Otherwise connectivity cannot be guaranteed. Thus departure can be divided in two phases 1) **trying phase**: At the end of this phase a process wishing to depart ensures that none of its neighbors are currently departing 2) **leaving phase**: In this phase, the process actually departs. It departs after connecting all its neighbors.

In a system where perpetual connectivity is maintained, consider a process p that wishes to depart and is in **trying phase**. Hence a process p can enter **leaving phase** only after ensuring that none of its current neighbors are also in **leaving phase**. To ensure this process p must contend with all its current neighbors to make sure that none of them enter the **leaving phase**. Thus this problem is similar to generalized version of dining philosopher's problem introduced in [2]. Processes in their **trying phase** can be viewed as processes waiting to enter critical section. And **leaving phase** can be considered as the critical section. But unlike traditional dining philosopher's problem the set of neighbors are continuously changing. This can be modeled as *dynamic dining philosopher's problem* [10].

From the above discussion we can see that a process wishing to depart may have to wait on some of its neighbors before it can depart. Thus as we can see if we wish to ensure perpetual connectivity then the departure latency of a process increases (whereas it is zero in case of eventual connectivity). We define a metric *depart-latency* as the time taken for a process to depart from the time it first wishes to. For measuring this metric we assume that message delays are nil and **leaving phase** takes one time unit. We believe that depart-latency is an important metric and a low depart-latency is highly desirable for many applications.

Many solutions have been proposed to the traditional static version of dining philosopher's problem. The most efficient of them being [6] which achieves $O(d)$ detection latency, where d is the maximum degree of a node. For measuring this latency the time taken to execute a critical section is assumed to be one time unit and message delays are assumed to be nil.

This motivates us to investigate if the same can be achieved for depart-latency in dynamic systems. Given a dynamic system, which maintains a topology such that the degree of no process ever exceeds d. Then can there be any algorithm that achieves a depart-latency of order $O(d)$?

Let us look at the depart-latency for the existing algorithms in dynamic systems. For comparing various algorithms we assume that the system starts with n processes and no new process joins the system. Only process departures are allowed. Under these assumptions, Baldoni et al's algorithm [1] and Peri and Mittal's algorithm [7] have worst case depart-latency of $O(n)$. Both these algorithms maintain a spanning tree. For Li et al's algorithm [4] worst case depart-latency is unbounded. To the best of our knowledge, we don't

know any algorithm that has depart-latency of $o(n)$ under these assumptions even when d is small.

3 Lower Bound

In [8] we show that the worst case depart-latency for any depart algorithms cannot be $o(\log(n))$ (where n is defined as above). We prove it by considering a set of processes which are arranged in the form of a chain. Any connected topology will have chain as its sub-structure. For instance, chain is a sub-structure of tree as well as a ring.

We shall briefly describe the idea. We prove this bound for a synchronous system (any bound for synchronous system will also hold for asynchronous system). We start with a chain of processes of length n. Assume that no new process joins the system. Suppose all the processes in the chain (and the system) wish to depart in the same round. The fastest way to achieve the departure of these n processes is as follows: in the first round $n/2$ processes depart (because two neighboring processes can not depart at the same time). After their departures the remaining re-organize themselves back as a chain. In the next round, $n/4$ processes depart and following run $n/8$ processes depart. Thus in this way we can see that there will be at least one process that has to wait for $O(log(n))$ time units before it can to depart.

This shows us that the worst case depart-latency for dynamic systems does not just depend on the degree of a node but also other nodes in the system. Next we would like to explore to see if there exists any algorithm that achieves worst case $o(n)$ depart-latency?

References

[1] R. Baldoni, J. Helary, and S. Tucci Piergiovanni. Maintaining group connectivity in dynamic asynchronous distributed systems. Technical report, "Dipartimento di Informatica e Sistemistica, A.Ruberti, Universit di Roma la Sapienza", 2004.

[2] K. M. Chandy and J. Misra. The Drinking Philosophers Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(4):632–646, 1984.

[3] D. Darling and J. Mayo. Stable Predicate Detection in Dynamic Systems. *Submitted to the Journal of Parallel and Distributed Computing (JPDC)*, 2003.

[4] X. Li, J. Misra, and C. G. Plaxton. Active and Concurrent Topology Maintenance. In *Proceedings of theSymposium on Distributed Computing (DISC)*, 2004.

[5] Xiaozhou Li and C. Greg Plaxton. On name resolution in peer-to-peer networks. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 82–89, New York, NY, USA, 2002. ACM Press.

[6] I. Page, T. Jacob, and E. Chern. Fast algorithms for distributed resource allocation. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):188–197, 1993.

[7] Sathya Peri and Neeraj Mittal. Monitoring stable properties in dynamic peer-to-peer distributed systems. In *Proceedings of the 25th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 420–431, Hyderabad, India, December 2005.

[8] Sathya Peri and Neeraj Mittal. Lower bounds on departures for dynamic asynchronous peer-to-peer systems. Technical report, University of Texas at Dallas, 2006.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[10] Weidman.E.B, Page.I.P, and Pervin.W.J. Explicit dynamic exclusion algorithm. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 142–149. IEEE, DEC 1991.