Separation of Voronoi Areas

Jonathan Sharyari

15th January 2013

Abstract

In 1985, Suzuki and Kazami presented a distributed algorithm for mutual exclusion. In this paper (?), the claim of mutual exclusion, deadlock and starvation freedom are tested using model checking techniques. The conclusion...

1 Introduction

1.1 Suzuki and Kazami's algorithm

The algorithm works as follows. Each of the N computer nodes, between which mutual exclusion is to be realized, runs two processes P1 and P2. Each node has two local arrays RN and LN of size N, and a queue Q. The array RN holds the latest received request identifier for each other node. Similarly, LN holds the request identifier of the latest carried out request of all the other nodes, of which the node has been informed. Since this is a distributed algorithm, these local arrays RN and LN are not the same for each node, since they are not always up-to-date . One reason is that this would require more message exchanges, but also that the messages are asynchronously received (??). The local queue Q holds the nodes that are currently requesting the privilege, in the order the requests have been received (first come, first serve ??).

When a node $node_i$ wants to enter its critical section, it first need to have the privilege to do so. In case the node already holds the privilige, it enters the critical section directly, otherwise a REQUEST (i, n) is sent to all nodes excluding itself, where i is the nodes identifier (index) and n is the request identifier, which is increased by one for each sent request.

Each node has a process P2 which receives the REQUEST(i,n) message from $node_i$. The array RN is updated, so that the corresponding entry RN[i] is set to n. If the node currently holding the privilege is not itself waiting for the privilege, it will forward the privilege to $node_i$ by sending a PRIVILEGE(Q, LN) message.

When $node_i$ receives a PRIVILEGE message, it can continue to carry out its critical work. When it is done, it updates its queue by appending all the nodes requesting, that are not already in the queue. Then, a PRIVILEGE is sent to the node first in Q, and that node is removed from the queue.

2 Problem formulation

3 Background

4 Algorithms

4.1 General outline

5 Results

6 Discussion

7 references

1. Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin
   CGAL manual chapter 20, 2D Arrangements

http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Arrangement_2/Chapter_main.html

2. O. Aichholzer, R. Fabila-Monroy, T. Hackl, M. van Kreveld, A. Pilz, P. Ramos, und B. Vogtenhuber

   Blocking Delaunay Triangulations.

   Proc. Canadian Conference on Computational Geometry, CCCG 2010, Winnipeg, August 9/11, 2010.

7.1 images

- Delaunay circumcircles, GNU Free Documenation Licence, Nü es

  http://commons.wikimedia.org/wiki/File:Delaunay_circumcircles.png

- Delaunay triangulation 1-3, public domain, user Capheiden

  http://upload.wikimedia.org/wikipedia/de/1/17/Voronoi-Delaunay.svg

  http://upload.wikimedia.org/wikipedia/de/4/48/Voronoi-Diagramm.svg

  http://upload.wikimedia.org/wikipedia/commons/1/1f/Delaunay-Triangulation.svg

8 Appendix A

```
1   const I: Integer;  (* the identifier of this node *)
2     var HavePrivilege, Requesting:  bool;
3     j, n:  integer;
4     Q:   queue of integer;
5     RN, LN:  array[l . . N] of integer;
6
7     (* The initial values of the variables are:
8     HavePrivilege = true in node 1, false in all other nodes;
9     Requesting = false;
10    Q = empty;
11    RN[j] = −1, j = 1,2, . . . , N;
12    LN[j] = −1, j = 1,2,. . . , N; *)
13
14    procedure Pl;
15    begin
16      Requesting := true;
17      if not HavePrivilege then
18      begin
19        RN[I] := RN[Z] + 1;
20        for all j in 11, 2, . . . , NJ − {Z) do
21          Send REQUEST(1, RN[I]) to node j;
22        Wait until PRIVILEGE(Q, LN) is received;
23        HavePrivilege := true
24      end,
25
26      Critical Section;
27
28      LN[Z] := RN[Z];
29      for all j in 11, 2, . . . , N) − [I) do
30        if not in(Q, j) and (RN[jJ = LN[j] + 1) then
31          Q := append(Q, j);
32      if Q f empty then
33      begin
34        HavePrivilege := false;
35        Send PRIVILEGE(tail(Q), LN) to node head(Q)
36      end;
37      Requesting := false
38    end,
39
40    procedure P2; (* REQUEST(j,n) is received; P2 is indivisible *)
41    begin
42      RN[j] := max(RN[j],n);
43      if HavePrivilege and not Requesting and (RN[j] = LN[j] + 1) then
44      begin
45        HavePrivilege := false;
46        Send PRIVILEGE(Q, LN) to node j
47      end
```

```
48      end,
```

Listing 1: Suzuki and Kazami's algorithm