Distributed Systems

Chang and Roberts algorithm

Raport

Patryk Rogalski

Cracow
20 January 2025

1 Introduction

Our problem is to find a leader in a certain distributed network. In our considerations, we will only examine structures with the following assumptions:

- Each process in the network has a unique identifier (ID)
- The goal is to elect the process with the highest ID as the leader
- Processes communicate asynchronously — messages may be delayed arbitrarily; there is no global clock, meaning processes operate independently and at their own pace
- The network topology is a directed ring
- In the end, each process knows whether it is:
  - A leader — the process with the highest ID
  - A subordinate — a process that is not the leader
  - Who the leader is *(this is not always required))*

2 Algorithm

2.1 Communication Protocol

Let us begin by formally describing our structure. We have a graph G with n vertices, where each vertex represents a process P_i, where i is the value in that vertex. These processes will communicate using the following message:
$MESSAGE(a, b)$ will contain two values:

- a - the ID of the process
- b - a flag (0 or 1) indicating whether the message is propagating the leader

Additionally, $P_{right(i)}$ denotes the right neighbor, and $P_{left(i)}$ the left neighbor of process i.

2.2 Execution Scheme

Initially, each process P_i sends a message $MESSAGE(i, 0)$ to its right neighbor $P_{right(i)}$.
Then we analyze the behavior of a single process P_x when it receives the message MESSAGE(y, b) from its left neighbor $P_{left(i)}$:

- if $b == 0$, we are still searching for the leader:
  - if $y < x$ the message is discarded
  - if $y > x$, a message $MESSAGE(y, 0)$ is sent to $P_{right(x)}$ and P_x knows it is a subordinate
  - if $y == x$, P_x knows it is the leader. If it is required that each node knows who the leader is, the leader starts propagating the message $MESSAGE(x, 1)$
- if $b == 1$
  - if $y \neq x$, the message $MESSAGE(y, 1)$ is forwarded
  - if $y == x$, the message has completed its round and is discarded

2.3 Conclusions

Note that the worst-case running time is equal to $n\times$ the time to send a single message. The pessimistic complexity of the total number of messages sent is $O(n^2)$. In the average case, the number of all messages sent is $O(n \log n)$.

2.4 Pessimistic Complexity

The algorithm can have this complexity when the processes are arranged in order $P_1, P_2, \ldots, P_n$, and then the i-th process must traverse a path of length i, so:

$$\sum_{i=1}^{n} i = \frac{n(n-1)}{2} = O(n^2)$$

2.5 Average Complexity

To calculate the average complexity, we need to compute the expected number of all messages, where E_i is the expected length of the path a message from node i will travel before it stops being propagated:

$$E = \sum_{i=1}^{n} E_i$$

We now compute the expected value for a specific node i:

$$E_i = \frac{n}{n+1-i}$$

Node i is the $(n+1-i)$-th largest node, so the probability that the message survives one step is that it encounters a smaller value, i.e.,:

$$\frac{n-(n+1-i)}{n} = \frac{i-1}{n}$$

Thus, the probability of stopping is:

$$\frac{n+1-i}{n}$$

What we are doing is making independent attempts to pass the ID to the next process. In each attempt, there is a chance of success, which is the ID not being passed further, and this probability is

$$\frac{n+1-i}{n}$$

What we are asking is: how many attempts are needed until the first success occurs? What we have here is a geometric distribution with a success probability of $p = \dfrac{n+1-i}{n}$, therefore:

$$E_i = \frac{1}{p} = \frac{1}{\frac{n+1-i}{n}} = \frac{n}{n+1-i}$$

Now we sum over all:

$$E = \sum_{i=1}^{n} E_i = \sum_{i=1}^{n} \frac{n}{n+1-i} = \sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n$$

Now let's estimate H_n. We'll use the fact that the function $f(x) = \dfrac{1}{x}$ is continuous and decreasing:

$$\int_1^{\infty} \frac{1}{x}\,dx = \lim_{n\to\infty} \int_1^{n} \frac{1}{x} dx = \lim_{n\to\infty} \ln x|_n^1 = \lim_{n\to\infty} (\ln n - \ln 1 + c) = \ln n + c$$

So

$$H_n = \ln n + O(1) \approx .69 \log n + O(1)$$

Therefore:

$$nH_n \approx .69n \log n + O(n) = O(n \log n)$$

3 Algorithm Minimizing Pessimistic Complexity

We can improve our algorithm by assigning each process P_i a random value $rand(i)$.

3.1 Communication Protocol

This time each process P_i additionally holds a value $rand(i)$, so the message will look like this: $MESSAGE(a, b, c)$ will contain three values:

- a - the randID of the process
- b - the ID of the process
- c - a flag 0, 1, or 2 depending on whether the message is searching the randID leader, propagating the randID leader, ID leader

3.2 Execution Scheme

We start by assigning each process P_i a random value $rand(i)$.
Then, initially, each process P_i sends the message $MESSAGE(rand(i), i, 0)$ to its right neighbor $P_{right(i)}$.

First, the search is for the Rand-leader. Once found, it starts propagating its info while also searching for the leader based on ID, and then propagating that information.
Now consider the behavior of a specific process P_x that receives the message $MESSAGE(u, y, b)$ form $P_{left(x)}$:

- if $b == 0$, we are still looking for the Rand-leader:
  - if $u < rand(x)$, the message is discarded
  - if $u > rand(x)$, the message $MESSAGE(u, x, 0)$ is forwarded to $P_{right(x)}$
  - if $u == rand(x)$, P_x knows it is the Rand-leader. It begins propagating the message $MESSAGE(u, y, 1)$ to find the ID leader
- if $b == 1$
  - if $u \neq rand(x)$
    - if $x > y$, update the ID and send $MESSAGE(u, x, 1)$
    - otherwise, forward the message $MESSAGE(u, y, b)$
  - if $u == rand(x)$, the message has returned to the Rand-leader and contains the leader by ID. It now starts propagating this information with the message $MESSAGE(u, y, 2)$
- if $b == 2$
  - if $u \neq rand(x)$, the process P_x knows it is a subordinate of y and forwards the info
  - if $u == rand(x)$, the message has returned to the Rand-leader and the communication ends

3.3 Conclusions

Since each process has an assigned random ID, we minimize the chance of encountering the worst-case scenario. The algorithm still has an average complexity of $O(n \log n)$, because propagation of messages with flags 1 and 2 is linear.

4 Code

```go
package king
import (
  "encoding/binary"
  "math/rand"
)

type ICandidate interface {
  SelectLeader() int
}

func NewCandidate(id int, input <-chan []byte, output chan<- []byte) ICandidate {
  x := rand.Uint64()
  //println("node", id, x)
  return &Candidate{id,input,output,x}
}

type Candidate struct {
  id int
  input <-chan[]byte
  output chan<- []byte
  randid uint64
}

func (candidate *Candidate) SelectLeader() int {

  data:= make([]byte,8*3)
    binary.BigEndian.PutUint64(data[0:8], candidate.randid)
    binary.BigEndian.PutUint64(data[8:16], 0)
    binary.BigEndian.PutUint64(data[16:24], uint64(candidate.id))
    go func(){
      //println("wysylka",candidate.id,candidate.randid,0,candidate.id)
      candidate.output <- data

    }()

  for {
    datarecive := <- candidate.input
    reciveId:= binary.BigEndian.Uint64(datarecive[0:8])
    ifleader:= binary.BigEndian.Uint64(datarecive[8:16])
    goodid:= binary.BigEndian.Uint64(datarecive[16:24])
    //println("odbiurka", candidate.id,reciveId,ifleader,goodid)
    if(ifleader==2){
      if(candidate.randid == reciveId){
        return int(goodid)
      } else{
        //println("wysylka",candidate.id,reciveId,ifleader,goodid)
        candidate.output <- datarecive
        return int(goodid)
      }
    }
    if(ifleader==1){
      if(candidate.randid == reciveId){
        code:=make([]byte,8*3)
        binary.BigEndian.PutUint64(code[0:8], reciveId)
```

```
    binary.BigEndian.PutUint64(code[8:16], 2)
    binary.BigEndian.PutUint64(code[16:24], goodid)
    //println("wysylka",candidate.id,reciveId,2,goodid)
     candidate.output <- code

   } else{
    if(candidate.id>int(goodid)){
     code:=make([]byte,8*3)
     binary.BigEndian.PutUint64(code[0:8], reciveId)
     binary.BigEndian.PutUint64(code[8:16], 1)
     binary.BigEndian.PutUint64(code[16:24], uint64(candidate.id))
     //println("wysylka",candidate.id,reciveId,1,candidate.id)
       candidate.output <- code

    } else {
     //println("wysylka",candidate.id,reciveId,1,goodid)
      candidate.output <- datarecive

    }
   }
  } else if(ifleader==0){
   if(candidate.randid < reciveId){
    code:=make([]byte,8*3)
    binary.BigEndian.PutUint64(code[0:8], reciveId)
    binary.BigEndian.PutUint64(code[8:16], 0)
    binary.BigEndian.PutUint64(code[16:24], uint64(candidate.id))
    //println("wysylka",candidate.id,reciveId,0,candidate.id)
     candidate.output <- code

   } else if( candidate.randid == reciveId){
    code:=make([]byte,8*3)
    binary.BigEndian.PutUint64(code[0:8], reciveId)
    binary.BigEndian.PutUint64(code[8:16], 1)
    binary.BigEndian.PutUint64(code[16:24], uint64(candidate.id))
    //println("wysylka kurwa",candidate.id,reciveId,1,uint64(candidate.id))
     candidate.output <- code

   }
  }
 }

}
```