TP2: Algoritmos Distribuídos de Eleição

Rafaela V. Albuquerque,*Virgínia Soares Müller†
- Faculdade Politécnica — PUCRS

04 de Novembro de 2024

Introdução

Este trabalho tem como objetivo apresentar uma implementação algorítmica para o problema de eleição em sistemas distribuídos, utilizando a topologia em anel e a linguagem de programação Golang. Nesse contexto, a eleição de um coordenador, ou líder, entre os processos participantes é fundamental para garantir a coordenação e o consenso em decisões, como atribuição de tarefas e controle de atividades. Em ambientes distribuídos, onde não existe um ponto central de controle, a seleção de um líder é crucial para assegurar a consistência e a eficácia das operações. Por exemplo, em uma rede de sensores utilizados para monitoramento ambiental, cada sensor coleta dados de temperatura, umidade ou qualidade do ar. Para otimizar a transmissão de dados e evitar sobrecarga na rede, um líder é necessário para coordenar a coleta e o envio de informações, garantindo que os dados sejam agregados de maneira eficiente e que a rede opere de forma harmoniosa, minimizando conflitos e melhorando a confiabilidade das informações coletadas.

Algoritmo de Eleição em Anel

O algoritmo de eleição em anel é uma abordagem eficiente utilizada em sistemas distribuídos para assegurar que um único processo seja designado como líder, encarregado de coordenar as atividades do sistema. Essa técnica é especialmente importante em ambientes onde múltiplos processos ou nós precisam colaborar de maneira coordenada, mas não há uma hierarquia fixa ou um servidor central que administre as operações. A seleção de um líder é vital para evitar conflitos e garantir que as decisões sejam tomadas de forma ordenada e eficaz. No algoritmo de eleição em anel, os processos estão dispostos em uma topologia circular, onde cada processo se comunica diretamente apenas com seus vizinhos imediatos. Quando uma eleição é iniciada, os processos trocam mensagens para determinar quem será o novo líder, baseando-se em critérios predefinidos, como o identificador do processo. Além disso, o algoritmo incorpora mecanismos para lidar com falhas de processos, permitindo que o sistema continue operando corretamente mesmo diante de falhas. Isso é feito por meio da simulação de falhas e recuperações, onde processos podem ser considerados inativos e, em seguida, restaurados, testando assim a robustez do algoritmo em cenários adversos. A utilização de canais e goroutines na linguagem Go possibilita uma implementação eficiente e concorrente do algoritmo, evidenciando a capacidade da linguagem para resolver problemas complexos de coordenação em sistemas distribuídos.

Implementação

A implementação do algoritmo de eleição em anel foi realizada com um conjunto de processos que se comunicam de forma eficiente através de canais, com o objetivo de eleger um líder entre eles. Utilizando a linguagem Go, o código é dividido em duas funções principais: ElectionController e ElectionStage. A função ElectionController desempenha um papel crucial ao gerenciar o ciclo de vida dos processos, simulando falhas e recuperações, além de iniciar o processo de eleição. Ela utiliza um canal de controle para enviar mensagens aos processos, informando sobre falhas, ativações e a necessidade de iniciar uma nova eleição, permitindo um controle centralizado sobre o fluxo de mensagens.
Cada processo no anel é representado pela função ElectionStage, que é responsável por receber e responder a mensagens conforme o protocolo de eleição. Quando uma eleição é iniciada (mensagem do tipo 1), o processo se candidata a líder e participa do processo de votação. Nesse contexto, cada processo envia seu identificador e coleta os votos dos outros participantes, garantindo que o processo de eleição siga as regras estabelecidas. O algoritmo inclui mecanismos para lidar com falhas: quando um processo falha (mensagem do tipo 2), ele altera seu estado para "falho"e não participa mais das eleições até ser restaurado. Por outro lado, se um processo for recuperado (mensagem do tipo 3), ele volta a participar do ciclo de eleição e votação, assegurando que o sistema mantenha sua funcionalidade.
A estrutura do código também reflete a necessidade de um gerenciamento eficiente do fluxo de mensagens. Os processos e o controlador são iniciados em goroutines, o que permite a execução concorrente e a comunicação fluida entre eles. O uso de canais é essencial para facilitar a troca de mensagens entre processos, uma vez que cada processo comunica diretamente apenas com seus vizinhos imediatos no anel. O WaitGroup é utilizado para garantir que o programa aguarde a finalização de todas as goroutines antes de encerrar, evitando que o programa seja encerrado prematuramente enquanto as eleições estão em andamento.

Conclusões

A implementação do algoritmo de eleição em anel demonstrou sua eficácia na coordenação de processos em um sistema distribuído, utilizando a linguagem Go para gerenciar a comunicação entre múltiplos processos. Através de uma estrutura clara de mensagens e canais, o algoritmo assegurou que um único processo fosse eleito como líder, mesmo diante de falhas simuladas. A funcionalidade do controlador permitiu a simulação de diferentes cenários, incluindo falhas e recuperações, provando a resiliência do sistema em manter a continuidade das operações.
Os testes realizados mostraram que, ao iniciar eleições e alterar o estado dos processos, o sistema foi capaz de adaptar-se às mudanças, garantindo que um novo líder fosse eleito sempre que necessário. Essa robustez é fundamental em aplicações práticas, onde a confiabilidade e a coordenação são essenciais para o funcionamento adequado do sistema.
Além disso, a implementação ilustra como a utilização de goroutines e canais em Go pode facilitar a criação de soluções concorrentes e eficientes para problemas complexos de coordenação em sistemas distribuídos. Esta abordagem pode ser expandida para aplicações mais sofisticadas, reforçando a importância do algoritmo de eleição em anel no campo da computação distribuída. A pesquisa e implementação de algoritmos como este são cruciais para o desenvolvimento de sistemas resilientes que operam em ambientes dinâmicos e interconectados, como na Internet das Coisas e computação em nuvem.

```
   Anel de processos criado

   Processo controlador criado

Controle: mudar o processo 0 para falho
 3: recebi mensagem 2, [ 0, 0, 0, 0 ]
 3: falho true
 3: lider atual 3
Controle: confirmação -5
Controle: inicia a eleição
 0: recebi mensagem 1, [ 0, 0, 0, 0 ]
 0: Estamos iniciando uma eleição
 0: Me candidatei a líder
 1: recebi mensagem 4, [ 0, -1, -1, -1 ]
 1: Me candidatei a líder
 2: recebi mensagem 4, [ 0, 1, -1, -1 ]
 2: Me candidatei a líder
 3: recebi mensagem 4, [ 0, 1, 2, -1 ]
 3: estou falhado
 0: Recebi mensagem 4, [ 0, 1, 2, -1 ]
 0: Iniciando processo de contabilização de votos
 0: O 2 foi eleito
 1: recebi mensagem 5, [ 2, 0, 0, 0 ]
 1: O 2 foi eleito
 2: recebi mensagem 5, [ 2, 0, 0, 0 ]
 2: O 2 foi eleito
 3: recebi mensagem 5, [ 2, 0, 0, 0 ]
 3: estou falhado
 0: Recebi mensagem 5, [ 2, 0, 0, 0 ]
 0: novo líder 2
Controle: confirmação 5
```

```
Controle: mudar o processo 2 para falho
 2: recebi mensagem 2, [ 0, 0, 0, 0 ]
 2: falho true
 2: lider atual 2
Controle: confirmação -5
Controle: inicia a eleição
 0: recebi mensagem 1, [ 0, 0, 0, 0 ]
 0: Estamos iniciando uma eleição
 0: Me candidatei a líder
 1: recebi mensagem 4, [ 0, -1, -1, -1 ]
 1: Me candidatei a líder
 2: recebi mensagem 4, [ 0, 1, -1, -1 ]
 2: estou falhado
 3: recebi mensagem 4, [ 0, 1, -1, -1 ]
 3: estou falhado
 0: Recebi mensagem 4, [ 0, 1, -1, -1 ]
 0: Iniciando processo de contabilização de votos
 0: O 1 foi eleito
 1: recebi mensagem 5, [ 1, 0, 0, 0 ]
 1: O 1 foi eleito
 2: recebi mensagem 5, [ 1, 0, 0, 0 ]
 2: estou falhado
 3: recebi mensagem 5, [ 1, 0, 0, 0 ]
 3: estou falhado
 0: Recebi mensagem 5, [ 1, 0, 0, 0 ]
 0: novo líder 1
Controle: confirmação 5
```

```
Controle: mudar o processo 2 para não falho
 3: recebi mensagem 3, [ 0, 0, 0, 0 ]
 3: falho false
 3: lider atual 3
Controle: confirmação -5
Controle: inicia a eleição
 0: recebi mensagem 1, [ 0, 0, 0, 0 ]
 0: Estamos iniciando uma eleição
 0: Me candidatei a líder
 1: recebi mensagem 4, [ 0, -1, -1, -1 ]
 1: Me candidatei a líder
 2: recebi mensagem 4, [ 0, 1, -1, -1 ]
 2: estou falhado
 3: recebi mensagem 4, [ 0, 1, -1, -1 ]
 3: Me candidatei a líder
 0: Recebi mensagem 4, [ 0, 1, 3, -1 ]
 0: Iniciando processo de contabilização de votos
 0: O 3 foi eleito
 1: recebi mensagem 5, [ 3, 0, 0, 0 ]
 1: O 3 foi eleito
 2: recebi mensagem 5, [ 3, 0, 0, 0 ]
 2: estou falhado
 3: recebi mensagem 5, [ 3, 0, 0, 0 ]
 3: O 3 foi eleito
 0: Recebi mensagem 5, [ 3, 0, 0, 0 ]
 0: novo líder 3
Controle: confirmação 5
```

```
Controle: mudar o processo 3 para não falho
 2: recebi mensagem 3, [ 0, 0, 0, 0 ]
 2: falho false
 2: lider atual 2
Controle: confirmação -5
Controle: inicia a eleição
 0: recebi mensagem 1, [ 0, 0, 0, 0 ]
 0: Estamos iniciando uma eleição
 0: Me candidatei a líder
 1: recebi mensagem 4, [ 0, -1, -1, -1 ]
 1: Me candidatei a líder
 2: recebi mensagem 4, [ 0, 1, -1, -1 ]
 2: Me candidatei a líder
 3: recebi mensagem 4, [ 0, 1, 2, -1 ]
 3: Me candidatei a líder
 0: Recebi mensagem 4, [ 0, 1, 2, 3 ]
 0: Iniciando processo de contabilização de votos
 0: O 3 foi eleito
 1: recebi mensagem 5, [ 3, 0, 0, 0 ]
 1: O 3 foi eleito
 2: recebi mensagem 5, [ 3, 0, 0, 0 ]
 2: O 3 foi eleito
 3: recebi mensagem 5, [ 3, 0, 0, 0 ]
 3: O 3 foi eleito
 0: Recebi mensagem 5, [ 3, 0, 0, 0 ]
 0: novo líder 3
Controle: confirmação 5

   Processo controlador concluído
```

Código - Algoritmo em GO

```go
// NOME: RAFAELA V. ALBUQUERQUE E VIRGÍNIA S. MÜLLER
package main

import (
  "fmt"
  "sync"
)

type mensagem struct {
  tipo  int     // tipo da mensagem para fazer o controle do que fazer (eleição, confirmacao da eleicao)
  corpo [4]int // conteudo da mensagem para colocar os ids (usar um tamanho ocmpativel com o numero de
    processos no anel)
}

var (
  chans = []chan mensagem{ // vetor de canias para formar o anel de eleicao - chan[0], chan[1] and chan[2] ...
    make(chan mensagem),
    make(chan mensagem),
    make(chan mensagem),
    make(chan mensagem),
  }
  controle = make(chan int)
  wg       sync.WaitGroup // wg is used to wait for the program to finish
)

// ElectionController controla o processo de eleição
func ElectionController(in chan int) {
  defer wg.Done()

  var temp mensagem

  // Comandos para o anel iniciam aqui

  // Mudar o processo 3 (canal de entrada 2) para falho (mensagem tipo 2)
  fmt.Println("Controle: mudar o processo 3 para falho")
    temp.tipo = 2
  chans[2] <- temp
  fmt.Printf("Controle: mudar o processo 0 para falho\n")

  fmt.Printf("Controle: confirmação %d\n", <-in) // receber e imprimir confirmação

  // Iniciar uma eleição pelo processo 0 (entrada 3)
  fmt.Println("Controle: inicia a eleição")
  temp.tipo = 1
  chans[3] <- temp
  fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação

  // Mudar o processo 2 (canal de entrada 1) para falho (mensagem tipo 2)
  fmt.Println("\n\nControle: mudar o processo 2 para falho")
  temp.tipo = 2
  chans[1] <- temp
  fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação

  // Iniciar uma eleição pelo processo 0 (entrada 3)
  fmt.Println("Controle: inicia a eleição")
  temp.tipo = 1
  chans[3] <- temp
  fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação

  // Mudar o processo 3 (canal de entrada 2) para funcionando (mensagem tipo 3)
  fmt.Println("\n\nControle: mudar o processo 2 para não falho")
  temp.tipo = 3
  chans[2] <- temp
  fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação

  // Iniciar uma eleição pelo processo 0 (entrada 3)
  fmt.Println("Controle: inicia a eleição")
  temp.tipo = 1
  chans[3] <- temp
  fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação

  // Mudar o processo 3 (canal de entrada 2) para funcionando (mensagem tipo 3)
  fmt.Println("\n\nControle: mudar o processo 3 para não falho")
  temp.tipo = 3
  chans[1] <- temp
  fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação

  // Iniciar uma eleição pelo processo 0 (entrada 3)
  fmt.Println("Controle: inicia a eleição")
  temp.tipo = 1
```

```
80    chans[3] <- temp
81    fmt.Printf("Controle: confirmação %d\n", <-in) // Receber e imprimir confirmação
82
83    temp.tipo = 6
84    for i := 0; i < len(chans); i++ {
85       chans[i] <- temp
86    }
87
88    fmt.Println("\n    Processo controlador concluído\n")
89 }
90
91 // ElectionStage representa um processo no anel de eleição
92 func ElectionStage(TaskId int, in chan mensagem, out chan mensagem, leader int) {
93    defer wg.Done()
94
95    // variaveis locais que indicam se este processo é o lider e se esta ativo
96
97    var actualLeader int
98    var bFailed bool = false // todos inciam sem falha
99
100   actualLeader = leader // indicação do lider veio por parâmatro
101
102   for temp := range in { // Ler mensagem
103      fmt.Printf("%2d: recebi mensagem %d, [ %d, %d, %d, %d ]\n", TaskId, temp.tipo, temp.corpo[0], temp.corpo[1],
       temp.corpo[2] ,temp.corpo[3])
104
105      if temp.tipo == 6 {
106         break
107      }
108
109      if bFailed and (temp.tipo != 2 and temp.tipo != 3) {
110         fmt.Printf("%2d: estou falhado\n", TaskId)
111         out <- temp
112         continue
113      }
114
115      switch temp.tipo {
116      case 1: // Inicia uma eleição
117         {
118            fmt.Printf("%2d: Estamos iniciando uma eleição\n", TaskId)
119            var data mensagem
120            data.tipo = 4
121            data.corpo[0] = TaskId
122            fmt.Printf("%2d: Me candidatei a líder\n", TaskId)
123
124            for i := 1; i < len(data.corpo); i++ {
125               data.corpo[i] = -1
126            }
127            out <- data
128            data = <-in
129
130            fmt.Printf("%2d: Recebi mensagem %d, [ %d, %d, %d, %d ]\n", TaskId, data.tipo, data.corpo[0], data.corpo[1],
        data.corpo[2], data.corpo[3])
131            var newLeader int = -1
132
133            for i := 0; i < len(data.corpo); i++ {
134               if newLeader < data.corpo[i] {
135                  newLeader = data.corpo[i]
136               }
137               data.corpo[i] = 0
138            }
139
140            fmt.Printf("%2d: Iniciando processo de contabilização de votos\n", TaskId)
141
142            data.tipo = 5
143            data.corpo[0] = newLeader
144            actualLeader = newLeader
145
146            fmt.Printf("%2d: O %d foi eleito\n", TaskId, actualLeader)
147
148            out <- data
149            data = <-in
150
151            fmt.Printf("%2d: Recebi mensagem %d, [ %d, %d, %d, %d ]\n", TaskId, data.tipo, data.corpo[0], data.corpo[1],
        data.corpo[2], data.corpo[3])
152
153            fmt.Printf("%2d: novo líder %d\n", TaskId, data.corpo[0])
154
155            controle <- 5
156         }
157      case 2: // Comando Falhar
158         {
```

```
159             bFailed = true
160             fmt.Printf("%2d: falho %v \n", TaskId, bFailed)
161             fmt.Printf("%2d: lider atual %d\n", TaskId, actualLeader)
162             controle <- -5
163         }
164     case 3: // Comando Retornar
165         {
166             bFailed = false
167             fmt.Printf("%2d: falho %v \n", TaskId, bFailed)
168             fmt.Printf("%2d: lider atual %d\n", TaskId, actualLeader)
169             controle <- -5
170         }
171     case 4: // Comando Escolhe um Líder
172         {
173             for i <- 0; i < len(temp.corpo); i++ {
174                 if temp.corpo[i] == -1 {
175                     temp.corpo[i] = TaskId
176                     break
177                 }
178             }
179             fmt.Printf("%2d: Me candidatei a líder\n", TaskId)
180             out <- temp
181         }
182     case 5: // Comando Seta Líder
183         {
184             actualLeader = temp.corpo[0]
185             fmt.Printf("%2d: O %d foi eleito\n", TaskId, actualLeader)
186             out <- temp
187         }
188     default:
189         {
190             fmt.Printf("%2d: não conheço este tipo de mensagem\n", TaskId)
191             fmt.Printf("%2d: lider atual %d\n", TaskId, actualLeader)
192         }
193     }
194 }
195 fmt.Printf("%2d: terminei \n", TaskId)
196 }
197
198 func main() {
199 wg.Add(5) // Add a count of four, one for each goroutine
200
201 // criar os processo do anel de eleicao
202
203 go ElectionStage(0, chans[3], chans[0], 3) // Não é líder, é o processo 3
204 go ElectionStage(1, chans[0], chans[1], 3) // Não é líder, é o processo 3
205 go ElectionStage(2, chans[1], chans[2], 3) // Não é líder, é o processo 3
206 go ElectionStage(3, chans[2], chans[3], 3) // Este é o líder
207
208 fmt.Println("\n   Anel de processos criado")
209
210 // criar o processo controlador
211
212 go ElectionController(controle)
213
214 fmt.Println("\n   Processo controlador criado\n")
215
216 wg.Wait() // Esperar as goroutines terminarem
217 }
```

Listing 1: Código Go