Algoritmo Distribuído de Eleição Baseado em Anel Lógico

Gustavo Willian Martins da Silva e Lorenzo Duarte More

1 Introdução

Em sistemas distribuídos, é comum aplicações possuírem um processo especial que funciona como coordenador, desempenhando um papel central para todos outros processos. Tal processo executa tarefas que são diferentes das outras realizadas normalmente pelos outros, como verificar a ocorrência de deadlock distribuído; ou seja, esse processo é especial. Os processos ativos (isto é, os que estão funcionando normalmente) que não possuem essa função monitoram constantemente o coordenador para ver se este está respondendo de forma adequada.

Porém, os processos, inclusive o coordenador, falham de tempos em tempos – seja por um problema de rede, de hardware ou por qualquer outro motivo – e as aplicações dependentes do sistema acabam ficando comprometidas. Acontece que quando o coordenador falha, o sistema, como um todo, pode parar de funcionar, já que aquele desempenha um papel importante e fundamental para manter os outros processos distribuídos funcionando normalmente.

Nesse sentido, a fim de evitar que as aplicações parem de funcionar, surge o problema da eleição: é preciso eleger um processo ativo para desempenhar o papel de novo coordenador e manter o sistema. A pergunta que surge é como escolher esse novo líder? Existem diversos critérios que podem ser escolhidos para eleger um coordenador; alguns exemplos incluem a quantidade de memória disponível, a velocidade do processamento de informação e a capacidade de comunicação. Cada critério é escolhido de acordo com as necessidades da aplicação.

De qualquer forma, o critério escolhido, também chamado de prioridade, serve para resolver o problema de substituir o coordenador que tenha falhado. Uma aplicação possível é eleger um líder em um banco de dados relacional ou hierárquico (*MySQL* e *Apache ZooKeeper*, por exemplo) para ser responsável pelas operações de escrita no banco. Todas essas operações passam pelo líder, e só ele é quem escreve; assim, nenhuma operação é perdida, e o banco não fica corrompido. Além disso, existem diversos algoritmos capazes de resolver esse problema. Para este trabalho, o algoritmo solicitado foi o baseado em anel lógico.

2 Algoritmo

A modelagem do anel implica em uma estrutura na qual os processos se conectam e formam uma sequência lógica em que todos conhecem o anel, mas só mandam mensagem para o próximo ativo na sequência. Assim que o coordenador falhar, o processo que detectou deve iniciar uma eleição, colocar a sua prioridade (que é única) em uma mensagem e mandar para o próximo ativo, e assim por diante. Quando a mensagem voltar para quem iniciou a eleição, este deve verificar quem foi o eleito, de acordo com a maior prioridade, e informar o próximo ativo até que todos saibam quem é o novo líder e passem a monitorar o seu comportamento.

Neste trabalho, o anel foi implementado com um vetor de canais que possuem a *struct* "mensagem"; esta contém o tipo da mensagem (número inteiro) e o corpo (vetor de inteiros), para armazenar prioridades (no caso, *ids* dos processos). Para cada processo, há variáveis locais que informam quem é o líder e se o processo em questão encontra-se ativo ou não. O tipo indica o que deve ser feito: falhar, voltar à ativa/iniciar eleição, colocar prioridade no corpo da mensagem, indicar quem é o novo líder ou encerrar processo. Ainda, foi escolhido um processo controle para monitorar toda a dinâmica; sua função e implementação são explicados na próxima Seção.

Quando um processo falha, deve ficar inativo e informar ao controle que falhou. Todas as mensagens devem enviar alguma confirmação para o controle. Quando algum processo inicia uma eleição, por exemplo, o controle deve saber quem será o novo líder. Caso um processo volte à ativa, ele imediatamente inicia uma eleição com grandes chances de vencer, tendo em vista que já venceu anteriormente. Para a mensagem do tipo encerramento, todos os processos recebem a mensagem, informam o seu recebimento e terminam a execução, saindo do laço infinito, feito para simbolizar um sistema distribuído.

O processo que receber mensagem de eleição (tipo 3) coloca todas posições do vetor em -5 (para indicar processos inativos) e coloca a sua *id* na respectiva posição. Após, manda mensagem (tipo 4) para o próximo processo colocar a *id* no corpo. Quando essa mensagem do tipo 4 volta para quem iniciou, o vetor é varrido e escolhe-se o maior número. A mensagem do tipo 5 então é enviada com o *id* do novo líder. Vale lembrar que essas ações só ocorrem em processos ativos.

3 Introduzindo Falhas

Para simular um sistema distribuído, foi preciso introduzir um processo controle responsável por decidir quem recebe uma mensagem de falha, quem volta à ativa/inicia uma eleição e por encerrar cada um dos processos. Isso porque é difícil falhar os processos criados em máquina local. Além disso, a linguagem utilizada (*Go*) também conta com mecanismos para evitar tais falhas.

Dessa forma, os processos aguardam instruções do controle até que a instrução seja a de encerrar. O tipo da mensagem é enviada a determinado processo, e o controle consome uma mensagem de confirmação do que aconteceu (qual processo falhou ou qual é o novo líder, por exemplo). Falhar um processo implica a alteração da sua variável local (nesse caso, pelo próprio processo) e simula uma falha de um caso real, que simplesmente para de responder.

Para testar o algoritmo, o controle pediu para o processo 0 falhar, que era o líder. A confirmação da falha é enviada e pode ser conferida com o que é impresso na tela. Após, pede para o 1 fazer eleição. Agora, de acordo com a prioridade, o novo líder passa a ser o 3 (Figura 1). Esse processo é repetido, com o 3 falhando e o 2 chamando eleição e vencendo (Figura 2). Depois, o 0 é reativado e convoca eleição, mas não ganha porque sua prioridade é menor que a do 2, que segue o líder (Figura 3). Por fim, o 3 volta à ativa e vence a eleição (Figura 4). Após essa interação, todas as funcionalidades foram testadas. Nesse momento, o controle pede para todos processos encerrarem e o programa acaba (Figura 4).

```
Controle: mudar o processo 0 para falho
Processo  0: falhei, enviando mensagem pro controle
Controle: confirmação de quem falhou: 0

Controle: processo 1, convoque nova eleicao
Processo  1: recebi mensagem 3, [ 0, 0, 0, 0 ]
Processo  1: colocando meu id na mensagem: [ -5, 1, -5, -5 ]
Processo  2: recebi mensagem 4, [ -5, 1, -5, -5 ]
Processo  2: colocando meu id na mensagem: [ -5, 1, 2, -5 ]
Processo  3: recebi mensagem 4, [ -5, 1, 2, -5 ]
Processo  3: colocando meu id na mensagem: [ -5, 1, 2, 3 ]
Processo  0: recebi mensagem 4, [ -5, 1, 2, 3 ]
Processo  0:  Processo falho, pula
Processo  1: recebi mensagem 4, [ -5, 1, 2, 3 ]
Processo  1: recebi todas ids: [ -5, 1, 2, 3 ]
Processo  1: vencedor eleicao: processo 3
Processo  1: informando novo lider aos outros processos
Processo  1: lider atualizado 3
Processo  2: lider atualizado: 3
Processo  3: lider atualizado: 3
Processo  0: Processo falho, pula
Controle: lider atual: 3
```

Figura 1: Processo 0 falha

```
Controle: mudar o processo 3 para falho
Processo  3: falhei, enviando mensagem pro controle
Controle: confirmação de quem falhou: 3

Controle: processo 2, convoque nova eleicao
Processo  2: recebi mensagem 3, [ 0, 0, 0, 0 ]
Processo  2: colocando meu id na mensagem: [ -5, -5, 2, -5 ]
Processo  3: recebi mensagem 4, [ -5, -5, 2, -5 ]
Processo  3:  Processo falho, pula
Processo  0: recebi mensagem 4, [ -5, -5, 2, -5 ]
Processo  0:  Processo falho, pula
Processo  1: recebi mensagem 4, [ -5, -5, 2, -5 ]
Processo  1: colocando meu id na mensagem: [ -5, 1, 2, -5 ]
Processo  2: recebi mensagem 4, [ -5, 1, 2, -5 ]
Processo  2: recebi todas ids: [ -5, 1, 2, -5 ]
Processo  2: vencedor eleicao: processo 2
Processo  2: informando novo lider aos outros processos
Processo  2: lider atualizado 2
Processo  3: Processo falho, pula
Processo  0: Processo falho, pula
Processo  1: lider atualizado: 2
Controle: lider atual: 2
```

Figura 2: Processo 3 falha

```
Controle: ativar o processo 0
Processo  0: recebi mensagem 3, [ 0, 0, 0, 0 ]
Processo  0: estou ativo novamente, chamando nova eleicao
Processo  0: colocando meu id na mensagem: [ 0, -5, -5, -5 ]
Processo  1: recebi mensagem 4, [ 0, -5, -5, -5 ]
Processo  1: colocando meu id na mensagem: [ 0, 1, -5, -5 ]
Processo  2: recebi mensagem 4, [ 0, 1, -5, -5 ]
Processo  2: colocando meu id na mensagem: [ 0, 1, 2, -5 ]
Processo  3: recebi mensagem 4, [ 0, 1, 2, -5 ]
Processo  3:  Processo falho, pula
Processo  0: recebi mensagem 4, [ 0, 1, 2, -5 ]
Processo  0: recebi todas ids: [ 0, 1, 2, -5 ]
Processo  0: vencedor eleicao: processo 2
Processo  0: informando novo lider aos outros processos
Processo  0: lider atualizado 2
Processo  1: lider atualizado: 2
Processo  2: lider atualizado: 2
Processo  3: Processo falho, pula
Controle: lider atual: 2
```

Figura 3: Processo 0 é ativado

```
Controle: ativar o processo 3
Processo  3: recebi mensagem 3, [ 0, 0, 0, 0 ]
Processo  3: estou ativo novamente, chamando nova eleicao
Processo  3: colocando meu id na mensagem: [ -5, -5, -5, 3 ]
Processo  0: recebi mensagem 4, [ -5, -5, -5, 3 ]
Processo  0: colocando meu id na mensagem: [ 0, -5, -5, 3 ]
Processo  1: recebi mensagem 4, [ 0, -5, -5, 3 ]
Processo  1: colocando meu id na mensagem: [ 0, 1, -5, 3 ]
Processo  2: recebi mensagem 4, [ 0, 1, -5, 3 ]
Processo  2: colocando meu id na mensagem: [ 0, 1, 2, 3 ]
Processo  3: recebi mensagem 4, [ 0, 1, 2, 3 ]
Processo  3: recebi todas ids: [ 0, 1, 2, 3 ]
Processo  3: vencedor eleicao: processo 3
Processo  3: informando novo lider aos outros processos
Processo  3: lider atualizado 3
Processo  0: lider atualizado: 3
Processo  1: lider atualizado: 3
Processo  2: lider atualizado: 3
Controle: lider atual: 3

Controle: finalizando processos um a um
Processo  2: mensagem para finalizar recebida.
Processo  1: mensagem para finalizar recebida.
Processo  3: mensagem para finalizar recebida.
Processo  0: mensagem para finalizar recebida.
```

Figura 4: Proceso 3 é ativado

```
// LORENZO MORE
// GUSTAVO SILVA

package main

import (
"fmt"
"sync"
)

type mensagem struct {
tipo  int    // tipo da mensagem para fazer o controle do que fazer
corpo [4]int // conteudo da mensagem para colocar os IDs
}

var (
chans = []chan mensagem{ // vetor de canais para formar o anel de eleicao
make(chan mensagem),
make(chan mensagem),
make(chan mensagem),
make(chan mensagem),
}
controle = make(chan int)
wg       sync.WaitGroup // wg e usado para esperar o programa terminar
)

func ElectionControler(in chan int) {
defer wg.Done()

var temp mensagem

// comandos para o anel iniciam aqui

// mudar o processo 0 - canal de entrada 3 - para falho
temp.tipo = 2
chans[3] <- temp
fmt.Printf("\nControle: mudar o processo 0 para falho\n")
fmt.Printf("Controle: confirmacao de quem falhou: %d\n", <-in) // confirmacao

// pedindo para processo 1 convocar eleicao
temp.tipo = 3
chans[0] <- temp
fmt.Printf("\nControle: processo 1, convoque nova eleicao\n")
fmt.Printf("Controle: lider atual: %d\n", <-in) // confirmacao

// mudar o processo 3 - canal de entrada 2 - para falho
temp.tipo = 2
chans[2] <- temp
fmt.Printf("\nControle: mudar o processo 3 para falho\n")
fmt.Printf("Controle: confirmacao de quem falhou: %d\n", <-in) // confirmacao

// pedindo para processo 2 convocar eleicao
temp.tipo = 3
chans[1] <- temp
fmt.Printf("\nControle: processo 2, convoque nova eleicao\n")
fmt.Printf("Controle: lider atual: %d\n", <-in) // confirmacao

// ativando processo 0
temp.tipo = 3
chans[3] <- temp
fmt.Printf("\nControle: ativar o processo 0\n")
fmt.Printf("Controle: lider atual: %d\n", <-in) // confirmacao

// ativando processo 3
```

```
temp.tipo = 3
chans[2] <- temp
fmt.Printf("\nControle: ativar o processo 3\n")
fmt.Printf("Controle: lider atual: %d\n", <-in) // confirmacao

// indicando finalizacao dos processos mandando para processo 1
fmt.Printf("\nControle: finalizando processos um a um\n")
temp.tipo = 7
chans[0] <- temp
chans[2] <- temp
chans[3] <- temp
chans[1] <- temp
}

func ElectionStage(TaskId int, in chan mensagem, out chan mensagem, leader int) {
defer wg.Done()

// variaveis locais que indicam se este processo e o lider e se esta ativo

var actualLeader int
var bFailed bool = false // todos iniciam sem falha

actualLeader = leader // indicacao do lider que foi passada como parametro

for {
temp := <-in // ler mensagem
if temp.tipo == 3 || temp.tipo == 4 {
fmt.Printf("Processo %2d: recebi mensagem %d, [ %d, %d, %d, %d ]\n",

        TaskId, temp.tipo, temp.corpo[0], temp.corpo[1], temp.corpo[2], temp.corpo[3])
}

switch temp.tipo {
// torna falho
case 2:
{
bFailed = true
fmt.Printf("Processo %2d: falhei, enviando mensagem pro controle\n", TaskId)
//fmt.Printf("Processo %2d: lider atual %d\n", TaskId, leader)
leader = -5
controle <- TaskId
}
// volta como era antes
case 3:
{
if bFailed == true {
fmt.Printf("Processo %2d: estou ativo novamente, chamando nova eleicao\n", TaskId)
}
// processo ativo
bFailed = false
leader = -5
temp.tipo = 4
for i := range temp.corpo {
temp.corpo[i] = -5 // desativa todos
}
temp.corpo[TaskId] = TaskId
fmt.Printf("Processo %2d: colocando meu id na mensagem: [ %d, %d, %d, %d ]\n",

        TaskId, temp.corpo[0], temp.corpo[1], temp.corpo[2], temp.corpo[3])
out <- temp
}
```

```
    // colocar ID no corpo
    case 4:
{
if temp.corpo[TaskId] == TaskId {
// Percorre o array a partir do segundo elemento (indice 1) e compara cada elemento com o valor maximo
fmt.Printf("Processo %2d: recebi todas ids: [ %d, %d, %d, %d ]\n",

            TaskId, temp.corpo[0], temp.corpo[1], temp.corpo[2], temp.corpo[3])
for i := 0; i < len(temp.corpo); i++ {
if temp.corpo[i] > temp.corpo[0] {
temp.corpo[0] = temp.corpo[i]
}
}

leader = temp.corpo[0]
temp.corpo[1] = TaskId
temp.tipo = 5
fmt.Printf("Processo %2d: vencedor eleicao: processo %d \n", TaskId, leader)
fmt.Printf("Processo %2d: informando novo lider aos outros processos\n", TaskId)
fmt.Printf("Processo %2d: lider atualizado %d\n", TaskId, leader)
} else if bFailed == false {
temp.corpo[TaskId] = TaskId
fmt.Printf("Processo %2d: colocando meu id na mensagem: [ %d, %d, %d, %d ]\n",

            TaskId, temp.corpo[0], temp.corpo[1], temp.corpo[2], temp.corpo[3])
} else {
fmt.Printf("Processo %2d:  Processo falho, pula\n", TaskId)
}
out <- temp
}
// mandando ID do novo lider para todos
case 5:
{
if TaskId != temp.corpo[1] {
if bFailed == false {
leader = temp.corpo[0]
fmt.Printf("Processo %2d: lider atualizado: %d \n", TaskId, leader)
} else {
fmt.Printf("Processo %2d: Processo falho, pula\n", TaskId)
}
} else {
temp.tipo = 6
controle <- leader
}
out <- temp

}
case 6:
{
// parte para sair do laco
break
}

// terminando processos
case 7:
{
if bFailed == true {
fmt.Printf("Processo %2d ja foi encerrado.\n", TaskId)
return
} else {
fmt.Printf("Processo %2d: mensagem para finalizar recebida.\n", TaskId)
bFailed = true
return
}
}
```

```
    default:
{
fmt.Printf("Processo %2d: nao conheco este tipo de mensagem\n", TaskId)
fmt.Printf("Processo %2d: lider atual %d\n", TaskId, actualLeader)
}
}
}
fmt.Printf("%2d: terminei \n", TaskId)
}

func main() {

wg.Add(5) // Adicione uma contagem de quatro, um para cada goroutine

// criar os processos do anel de eleição

go ElectionStage(0, chans[3], chans[0], 0) // este e o lider
go ElectionStage(1, chans[0], chans[1], 0) // nao e lider, e o processo 0
go ElectionStage(2, chans[1], chans[2], 0) // nao e lider, e o processo 0
go ElectionStage(3, chans[2], chans[3], 0) // nao e lider, e o processo 0

fmt.Println("\n   Anel de processos criado")

// criar o processo controlador

go ElectionControler(controle)

fmt.Println("\n   Processo controlador criado")

wg.Wait() // Esperar pelas goroutines terminarem
}
```