Eleição de líder

Algoritmos de eleição de líder

- Muitos algoritmos distribuídos necessitam de seleccionar um processo de entre um conjunto de processos iguais: um coordenador ou um **líder**.

O que queremos assegurar:

- *Safety*: Todos os processos escolhem o mesmo líder.
  - Tipicamente: cada processo tem um *id,* o líder deve ser o processo com maior *id.*
- *Liveness*: A execução do algoritmo termina.

- Parece simples?...

Eleição de líder

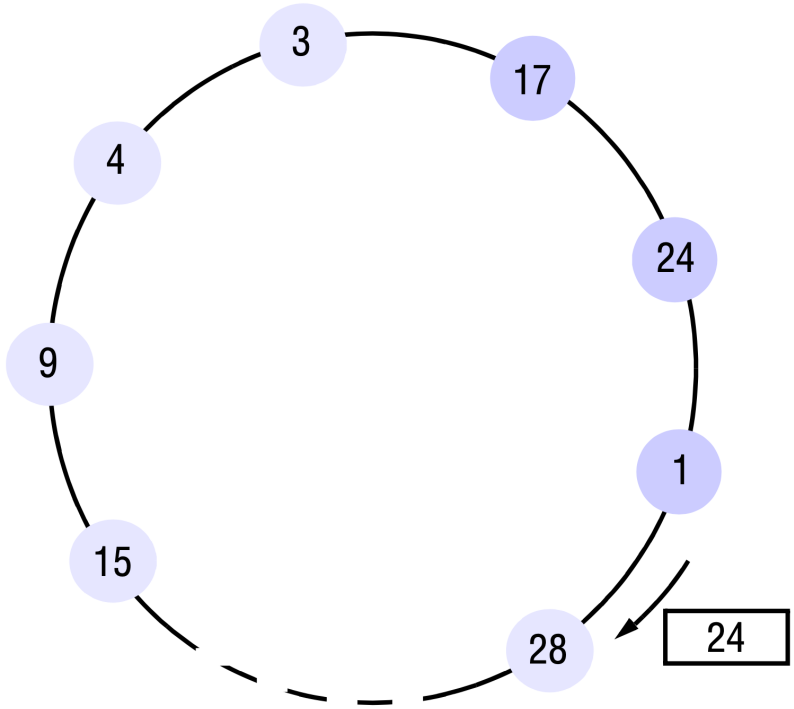
- Se não considerarmos falhas (ou seja, algoritmo de eleição só é executado após a falha do anterior líder ser detetada)
  - Quantas mensagens são necessárias para eleger um líder?

- Se considerarmos falhas
  - Para além de eleger um novo líder temos de detectar a falha do líder anterior

Complexidade de mensagens

- A complexidade de mensagens para aleição de um líder tem sido estudada para diferentes tipos de redes
- Um caso clássico é a eleição de um líder no caso em que os processos estão organizados numa rede em anel

Eleição num anel



Retirado da página 643 do livro da cadeira!

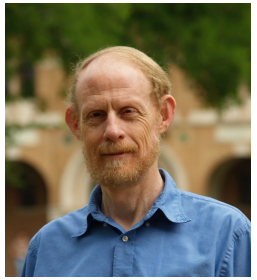
Eleição num anel: algoritmo

- Quando um processo p decide iniciar uma eleição:
  - Marca-se como participante
  - **Prepara uma mensagem *election(id(p))*** e envia-a ao próximo no anel
- Quando um processo p recebe uma mensagem *election(id):*
  - **Se o id na mensagem é superior ao identificador local**: p reencaminha-a ao próximo e p marca-se como participante.
  - **Se o id na mensagem é inferior** e p ainda não participava: **substitui** o id na mensagem pelo de p, reencaminha-a ao próximo e marca-se como participante.
  - **Se o id na mensagem é o de p**, então p **torna-se o novo líder**! Marca-se como não participante e **envia mensagem *elected(id(p))*** ao próximo no anel.
- Quando um processo p recebe uma mensagem *elected(id):*
  - Aprende que o novo líder é aquele indicado na mensagem, reencaminha a mensagem e marca-se como não participante.
  - Se o id na mensagem for p, não faz nada (o algoritmo terminou).

Eleição num anel: complexidade

- Este algoritmo, no caso em que só um processo dá início à eleição, pode gerar 3N-1 mensagens
  - Ver página 659 do Coulouris
- No entanto, quer o livro do Coulouris, quer o livro do van Steen não dão enfâse a um aspecto importante:
  - Se todos os processos começarem a eleição simultaneamente, o custo deste algoritmo é quadrático!
  - Será possível conceber um algoritmo em que o custo no pior caso não seja quadrático?

Reuzir a complexidade de mensagens



Technical Note
Operating Systems

R. Stockton Gaines
Editor

Decentralized Extrema-Finding in Circular Configurations of Processors

D.S. Hirschberg and J.B. Sinclair
Rice University

This note presents an efficient algorithm, requiring

pass messages in either or both directions, that these directions are distinguished, that processors can detect from which direction a received message originated, but that "left" may not mean the same to all processors. We propose an algorithm that requires $O(n \log n)$ messages in the worst case. The algorithm as given elects the processor with the highest value.

In the algorithm given below, a processor can initiate messages in both directions by a *sendboth* directive. A processor can pass a (possibly modified) message in a circular manner by a *sendpass* directive. A processor can send a responsive message back in the direction from which that processor received a message by a *sendecho* directive.

The Algorithm
To run for election:
    status ← "candidate"
    maxnum ← 1
    WHILE status = "candidate" DO
        sendboth ("from", myvalue, 0, maxnum)
        await both replies (but react to other messages)

Eleição em anel por torneio

- Os processos procuram um líder num horizonte que duplica em cada turno
- Em cada turno o número de competidores vai sendo reduzido para metade
- Isto resulta na execução de log(n) turnos para uma complexidade total de *n log(n)*

Algoritmo “Bully”

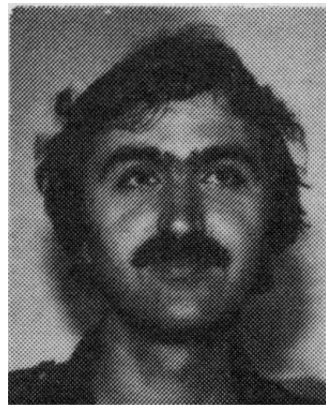
- Objetivo: eleger o processo com maior identificador (tal como antes)

- Pressupostos:
  - Há tempos máximos conhecidos para a comunicação; canais fiáveis;
  - Processos podem falhar (crash).
  - Cada processo conhece os identificadores dos outros.

Bully

Elections in a Distributed Computing System

HECTOR GARCIA-MOLINA, MEMBER, IEEE



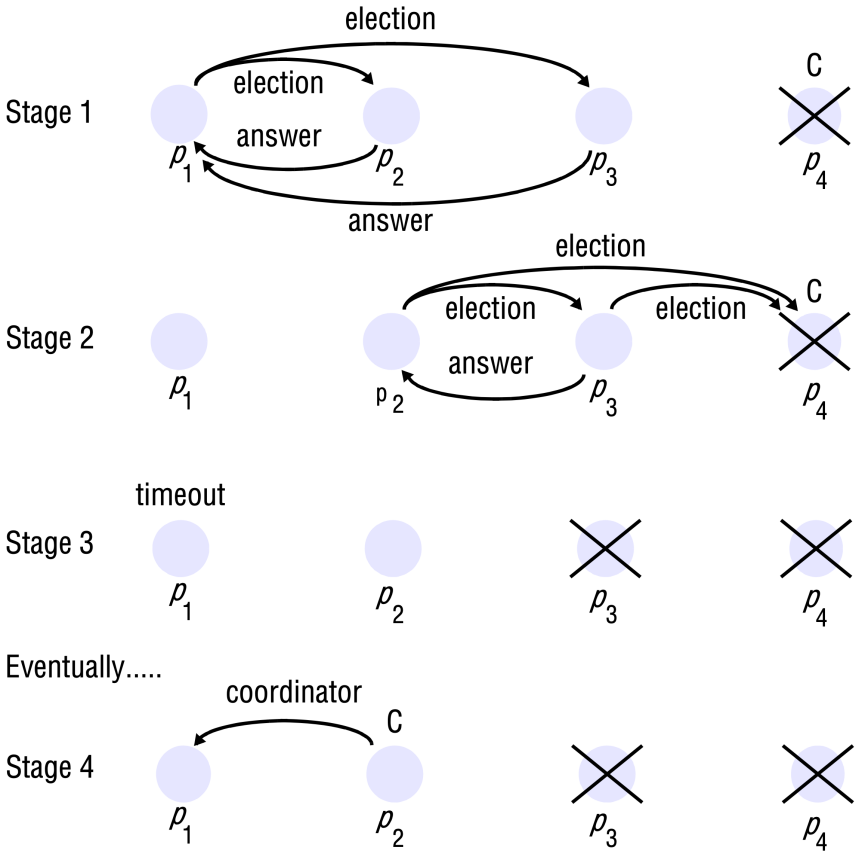
***Abstract*—After a failure occurs in a distributed computing system, it is often necessary to reorganize the active nodes so that they can continue to perform a useful task. The first step in such a reorganization or reconfiguration is to elect a coordinator node to manage the operation. This paper discusses such elections and reorganizations. Two types of reasonable failure environments are studied. For each environment assertions which define the meaning of an election are presented. An election algorithm which satisfies the assertions is presented for each environment.**

***Index Terms*—Crash recovery, distributed computing systems, elections, failures, mutual exclusion, reorganization.**

out to *reorganize* the system. During the reorganization period, the status of the system components can be evaluated, any pending work can either be finished or discarded, and new algorithms (and possibly a new task) that are tailored to the current situation can be selected. The reorganization of the system is managed by a *single* node called the *coordinator*. (Having more than one node attempting to reorganize will lead to serious confusion.) So as a first step in any reorganization, the operating or active nodes must *elect* a coordinator. It is precisely these elections we wish to study in this paper.

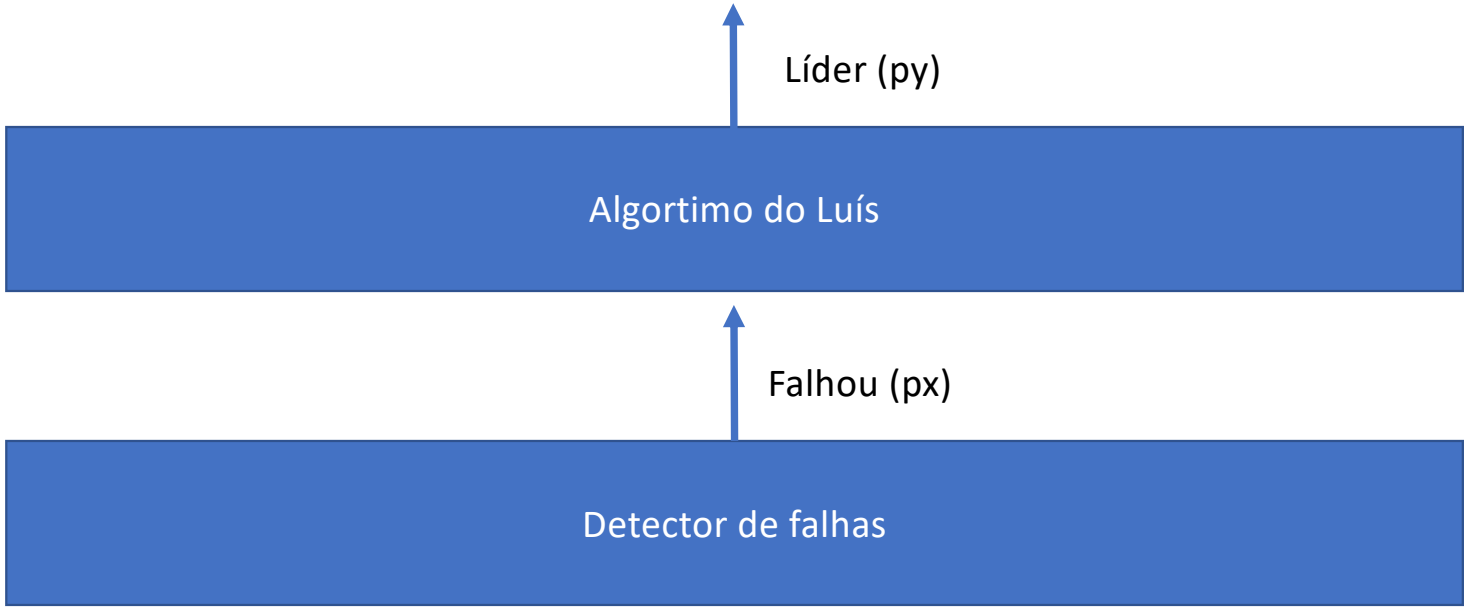
In this paper we will not study the first strategy of contin-

Algoritmo do “bully”



Retirado da página 644 do livro da cadeira!

Algoritmo do “Luís”



Algoritmo do “Luís”

- Init
  - activos = {p0, p1, p2, …, pn}
  - líder = maxid (activos)
  - Output (líder)
- Quando falhou (px)
  - activos = activos \ px
  - líder = maxid (activos)
  - Output (líder)

Retirado da cabeça do Luís!

Bully vs Luís

- Algoritmo do Luís
  - Simples
  - Modular
  - Menos eficiente pois obriga a detectar falhas de nós que não são candidatos a líder
- Algoritmo Bully
  - Mistura detecção de falhas com eleição de líder
  - Cada nó só tem de detectar a falha de nós com id superior ao seu

Qual o problema destes algoritmos?

- Assumem que a detecção de falhas é perfeita
- Desta forma, conseguem assegurar que existe sempre um único lider escolhido entre os nós que não falharam

Deteção de falhas perfeita

- A falha de um nó é sempre detectada
- Um nó correcto não é dado como falhado

- Mas isto pressupõe que…
  - O sistema é síncrono (isto é, a falha pode ser detectada com exactidão através do uso de temporizadores)
  - Não existem partições na rede

- E isto nem sempre são pressupostos realistas!

E se o detetor de falhas não for perfeito?

- Processos diferentes podem discordar sobre a identidade do líder
- Podem coexistir múltiplos líderes num dado instante

Idealmente…

- Desenvolver algoritmos tolerantes a falhas cuja correcção não dependesse de um detector de falhas perfeito

Detecção de falhas “alguma-vez” perfeita

- O detector de falhas pode, durante um tempo indeterminado, errar
  - Por exemplo, declarar um no falhado e posteriormente indicar que o nó afinal está activo
- Mas existe um instante a partir do qual o detector de falhas deixa de errar
- Um componente que estas caracerísticas é designado por um detector de falhas “alguma-vez” (do inglês “*eventually*”) perfeito
- Isto permite eleger um líder de forma “alguma-vez” perfeita

Diferentes tipos de problemas

- Existem problemas que se podem resolver sem necessitar de um detector de falhas
- Mas também existem problemas que só têm solução determinista se for possível constuir um detector de falhas ”alguma-vez” perfeito

Dois exemplos

Que estudaremos mais tarde

Difusão fiável

- Um processo pretende enviar uma mensagem para um grupo de processos
- Alguns processos podem falhar (incluindo o emissor). Os processos que não falham designam-se por correctos.
- No final, todos os processos correctos recebem a mensagem e nenhum processo correcto recebe a mensagem

- **Este problema pode ser resolvido sem necessitar de detecção de falhas**

Consenso

- Existem N processos cada um com um valor incial
- Alguns processos podem falhar
- No final todos os processo que não falham acordam no mesmo valor (entre os valores iniciais)

- **Este problema só tem solução determinista se for possível eleger um líder de forma “alguma-vez” perfeita**