

# Detector de Fallos

Ernesto Jiménez, Prometeo  
EPN

[ernes@etsisi.upm.es](mailto:ernes@etsisi.upm.es)

# Índice

- Elección de líder.
  - Algoritmo basado en anillo.
  - Algoritmo del dictador.
- Elección futura de líder.
  - Algoritmo con canales síncronos futuros.
  - Sincronía mínima.
  - Algoritmo con sincronía mínima.

# Bibliografía

- M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg. On implementing omega with weak reliability and synchrony assumptions. PODC 2003. 306-314.
- T. Chandra, V. Hadzilacos, S. Toueg. The Weakest Failure Detector for Solving Consensus. J. ACM 43(4). 685-722 (1996).
- T. Chandra, S. Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. J. ACM 43(2). 225-267 (1996).
- E.G. Chang and R. Roberts. An improved algorithm for decentralized extremafinding in circular configuration of processors. Communications of the ACM. 22 (5) 281-3, 1979.
- Antonio Fernández Anta. Elección futura de líder en sistemas distribuidos no fiables. Curso de doctorado. Universidad Rey Juan Carlos. 2006.
- Antonio Fernández, Ernesto Jiménez, Sergio Arévalo. Minimal system conditions to implement unreliable failure detectors. PODC 2005, 207 (versión extendida PRDC 2006. 63-72).
- Hector Garcia-Molina. Elections in Distributed Computer Systems. IEEE Transactions on Computers. C-31 (1) 48-59, 1982.

# Elección de Líder.

- Algoritmo en el que se elige un único proceso para adoptar un papel particular dentro de un protocolo (p.ej. secuenciador para el radiado ordenado totalmente).
- La elección de líder es iniciada a petición de algún proceso.
- Si se inician concurrentemente elecciones por parte de distintos procesos, sólo un líder debe ser elegido.
- Se elige el proceso con mayor identificador.
- Seguridad: un proceso participante tiene como valor de líder el identificador de un proceso vivo con el mayor identificador o un valor indefinido
- Viveza: todos los procesos eligen un líder o se caen.

## Elección de Líder. Algoritmo basado en anillo

- Sistema síncrono [Chang & Roberts79]. Cada proceso tiene un canal con el siguiente proceso en el anillo. Los mensajes circulan en sentido de las agujas del reloj.
- El proceso que inicia el algoritmo se marca como participante y envía su identificador en un mensaje de *elección* a su vecino.
- Cuando un proceso recibe un mensaje de elección compara el identificador recibido con el suyo.
  - Si es menor el recibido y el proceso no es un participante, sustituye el identificador en el mensaje por el suyo y lo reenvía al vecino y se marca como participante.

## Elección de Líder. Algoritmo basado en anillo (cont.)

- Si es mayor el recibido, reenvía el mensaje y se marca como participante.
- Si es menor el recibido y el proceso es un participante, no hace nada (no envía ningún mensaje).
- Si el identificador coincide con el del proceso, ese proceso es el líder.
- El líder se marca como no participante y envía un mensaje *elegido* al siguiente proceso.
- Cuando un proceso distinto al líder recibe este mensaje, anota qué proceso es el líder y reenvía el mensaje.

## Elección de Líder. Algoritmo basado en anillo (cont.)

- El estado participante no participante sirve para eliminar elecciones concurrentes lo antes posible.
- El peor caso es cuando el proceso con mayor identificador es el anterior al que inicia la elección de líder.  $3n-1$  mensajes.
  - Hacen falta  $n-1$  mensajes para alcanzar a ese proceso.
  - $n$  mensajes para transmitir el identificador de ese proceso.
  - $n$  mensajes de elegido.
- No tolera fallos de ningún proceso.

# Elección de Líder. Algoritmo del dictador

- Permite caídas de los procesos, supone canales fiables. Sistema síncrono [García-Molina 82].
- Supone que cada proceso conoce a los procesos con identificadores mayores y que se puede comunicar con esos procesos.
- Hay tres tipos de mensajes:
  - *Elección*, para anunciar una elección.
  - *Respuesta*, enviado en respuesta a un mensaje de elección.
  - *Coordinador*, anuncia la identidad del proceso líder.



## Elección de Líder. Algoritmo del dictador (cont.)

- Un proceso inicia el algoritmo cuando sospecha (por medio de *timeouts*) que el líder se ha caído.
- El proceso que sabe que tiene el mayor identificador puede enviar un mensaje de coordinador a todos los procesos con identificador menor para indicar que es el nuevo líder.
- Los procesos con menor identificador inician el algoritmo enviando un mensaje de elección a los procesos con mayor identificador y esperan un mensaje de respuesta.

## Elección de Líder. Algoritmo del dictador (cont.)

- Cuando a un proceso le llega un mensaje de elección envía una respuesta e inicia una elección, a no ser que ya la haya iniciado.
- Si a un proceso no le llega ningún mensaje de respuesta, se considera el nuevo coordinador y envía un mensaje de coordinador a los procesos con menor identificador.
- Si le llegan las respuestas, espera a que le llegue un mensaje de coordinador. Si éste no llega, inicia el algoritmo de nuevo.
- Cuando un proceso recibe un mensaje de coordinador anota la identidad del nuevo líder.

## Elección de Líder. Algoritmo del dictador (cont.)

- El algoritmo se llama así porque cuando un proceso reemplaza a un proceso caído, si tiene el mayor identificador, se impondrá como líder aunque el líder esté vivo.
- La propiedad de seguridad no se cumple si los procesos caídos se reemplazan con procesos con la misma identidad.
- El nuevo proceso y uno vivo enviarán mensajes de coordinador concurrentemente. Lo mismo ocurre si los timeouts no son precisos.
- En el peor de los casos hacen falta  $O(n^2)$  mensajes, el proceso con menor identificador detecta el fallo del líder y  $n-1$  procesos inician la elección.

## Elección futura de Líder.

Clase de detector de fallos  $\Omega$  (*eventual leader election*) [Chandra, Hadzilacos, Toueg 1996]:

- Cada proceso  $p$  tiene una variable  $leader_p$  con un id. de proceso.
- Terminación y acuerdo: A partir de un determinado momento, todo proceso correcto  $p$  tiene permanentemente  $leader_p = \ell$ , siendo  $\ell$  un proceso correcto.
- Es el detector de fallos más débil que permite resolver consenso en sistemas **asíncronos** ( $\Omega \equiv \langle \rangle W$ )

## Elección futura de Líder (cont.)

Comparación con elección de líder [García-Molina 1982]:

- Cada proceso  $p$  tiene una variable booleana  $leader_p \in \{\text{true}, \text{false}\}$ .
- Un proceso  $\ell$  es elegido como líder en el momento en que se pone  $leader_\ell = \text{true}$ .
- Acuerdo: Nunca hay dos líderes (procesos  $p$  y  $q$  con  $leader_p = leader_q = \text{true}$ ).
- Terminación: A partir de un determinado momento, algún proceso  $\ell$  acaba por establecer  $leader_\ell = \text{true}$ .

## Elección futura de Líder (cont)

- Elección de líder y  $\Omega$  no se pueden resolver en un sistema asíncrono con  $f > 0$ : Por el resultado de imposibilidad FLP.
- Elección de líder requiere un detector de fallos más fuerte que  $\Omega$ .

Hay que imponer alguna restricción adicional al sistema: **sistema parcialmente síncrono.**

# Elección futura de Líder. Algoritmo con canales síncronos futuros

Sistema distribuido **parcialmente síncrono** con paso de mensajes:

- Conjunto finito de  $n$  procesos  $\Pi = \{p_1, \dots, p_n\}$ .
- Fallos de parada (crash)
- Los procesos envían y reciben mensajes usando canales bidireccionales que unen cada par de procesos: *send(m)* y *receive(m)*
- La mínima y máxima velocidad de avance de los procesos están acotadas, pero las cotas no son conocidas.

# Elección futura de Líder. Algoritmo con canales síncronos futuros (cont.)

- Canal síncrono futuro (*eventually timely*): tiene dos parámetros  $T$  y  $\Delta$  desconocidos tal que:
  - Hasta el instante  $T$  el canal no da garantías. Todos los mensajes se pueden perder o llegar tarde.
  - A partir de  $T$ , un mensaje enviado en un instante  $t$  se entrega como muy tarde en instante  $t + \Delta$ .
- Todos los canales son síncronos futuros.
- $send(m)$ : si  $m$  es enviado por un proceso correcto a un proceso correcto a través de un canal síncrono futuro en un instante  $t > T$ ,  $m$  es recibido como tarde en instante  $t + \Delta$  (antes de  $t$  no hay garantías).
- Fiable es un caso particular de síncrono futuro (cuando  $T=0$  y  $\Delta$  es conocido).



# Elección futura de Líder. Algoritmo con canales síncronos futuros (cont.)

[Chandra & Toueg 1996]

## Initially:

$leader_p \leftarrow 0$

$tout_p \leftarrow \eta$

**set timer to**  $tout_p$

**start** T1 y T2

## Task T1:

**repeat** each  $\eta$  time

send (ALIVE) a todos

## Task T2:

when recibido mensaje ALIVE  
from  $q$  and  $q = leader_p$ :

**reset timer to**  $tout_p$

when recibido mensaje ALIVE  
from  $q$  and  $q < leader_p$ :

$leader_p \leftarrow q$

$tout_p \leftarrow tout_p + 1$

**reset timer to**  $tout_p$

when timer expira:

$leader_p \leftarrow leader_p + 1$

**set timer to**  $tout_p$

## Elección futura de Líder. Algoritmo con canales síncronos futuros (cont.)

Consideremos instante  $t > T$  en que todos los procesos que van a fallar lo han hecho:

- Los latidos del proceso correcto con identificador más pequeño,  $\ell$ , llegan como mucho cada  $\eta + \Delta$ .
- Si la variable  $tout_p$  no alcanza ese valor es que deja de incrementarse. Entonces, tras recibir un ALIVE de  $\ell$ ,  $leader_p = \ell$ .
- Si la variable alcanza ese valor, no se incrementa más, y tras recibir un ALIVE de  $\ell$ ,  $leader_p = \ell$ .

# Síncronía mínima.

Modelo de sistema con procesos parcialmente síncronos, canales síncronos futuros y asíncronos con pérdidas (sin garantías):

- **Propiedad 1.** *En cada ejecución, al menos un proceso correcto puede alcanzar los demás procesos correctos a través de caminos de canales síncronos futuros y procesos correctos.*
- *Para poder implementar un elector de tipo  $\Omega$ , un sistema con canales síncronos futuros y asíncronos con pérdidas debe satisfacer la Propiedad 1.*  
[Fernández et al 2006]

## Síncronía mínima (cont.)

- Supongamos que hay un algoritmo  $A$  que implementa el elector en un sistema más débil.
- Considera una ejecución  $R$  sin fallos en que todos los mensajes en canales con pérdidas se pierden:
  - Se elige a  $\ell$  como líder.
  - Algún proceso  $p$  no recibe nada de información de  $\ell$ .
- Considera una ejecución  $R'$  en que  $\ell$  falla pero que desde el punto de  $p$  es igual que  $R$ .  $p$  elige a  $\ell$  permanentemente como líder.

# Algoritmo con sincronía mínima.

[Aguilera et al 2003]

## Initially:

$leader_p \leftarrow p$   
 $suspect_p[i] \leftarrow 0, \forall i \in \Pi$   
**set** timer<sub>p</sub>[i] to  $\eta, \forall i \in \Pi$   
**start** T1 y T2

## Task T1:

**repeat** each  $\eta$  time  
    send (ALIVE,p,suspect<sub>p</sub>) a todos

## Task T2:

when mensaje (ALIVE,q,suspect<sub>q</sub>) es recibido por primera vez y  $q \neq p$ :

    send (ALIVE,suspect<sub>q</sub>) a todos

$suspect_p[k] \leftarrow \max(suspect_p[k], suspect_q[k]), \forall k \in \Pi$

$leader_p \leftarrow \min_k \{(suspect_p[k], k)\}$

**reset** timer<sub>p</sub>[q] to  $(\eta + suspect_p[q])$

when timer<sub>p</sub>[q] expira:

$suspect_p[q] \leftarrow suspect_p[q] + 1$

$leader_p \leftarrow \min_k \{(suspect_p[k], k)\}$

**set** timer<sub>p</sub>[q] to  $(\eta + suspect_p[q])$

# Algoritmo con sincronía mínima (cont.)

## Corrección del algoritmo:

- Lema 1. Dado un proceso correcto  $p$  y un proceso  $q$  que falla,  $\text{suspect}_p[q]$  crece sin cesar.
- Para cada proceso correcto  $p$ , definimos:  
$$V_p = \sup_t \{ \text{suspect}_p[p](t) \}$$
- Lema 2. Dado un proceso correcto  $p$  que satisface la Propiedad 1,  $V_p < \infty$ .

# Algoritmo con sincronía mínima (cont.)

Sea  $B = \{p : V_p < \infty\}$

- Lema 3. Hay un tiempo tras el cual, para todo proceso  $q \in B$  y todo proceso  $p$  que satisface Propiedad 1, se tiene  $\text{suspect}_p[q] = V_q$ .
- A partir de un instante,  $p$  debe recibir mensajes de  $q$  periódicamente, ya que  $\text{timer}_p[q]$  expira un número acotado de veces.
- El primero que reciba con  $\text{suspect}_q[q] = V_q$  establecerá el valor.
- Tras ese momento el valor no se modifica.

# Algoritmo con sincronía mínima (cont.)

Sea  $\ell = \min_p \{V_p, p\}$

- Lema 4. Hay un tiempo tras el cual, para todo proceso correcto  $q$  tal que  $q$  no está en  $B$ , y todo proceso  $p$  que satisface Propiedad 1 se tiene  $\text{suspect}_p[q] > V_\ell$ .
  - Una vez  $\text{suspect}_q[q] > V_\ell$  hay dos casos:
    1.  $p$  recibe algún mensaje de  $q$ . Entonces actualiza  $\text{suspect}_p[q] > V_\ell$ .
    2. Si no, expirará  $\text{timer}_p[q]$  hasta que  $\text{suspect}_p[q] > V_\ell$ .



# Algoritmo con sincronía mínima (cont.)

- Lema 5. Dado un proceso  $p$  que satisface Propiedad 1, el valor de  $\text{suspect}_p[q]$ ,  $\forall q$  es continuamente propagado a todos los procesos correctos.
- Como consecuencia,  $\ell$  es elegido líder por todos los procesos correctos a partir de un determinado momento.