Consenso

Ricardo Jiménez Peris, Marta Patiño Martínez y Ernesto Jiménez

Distributed
Systems
Laboratory
Universidad Politécnica de Madrid (UPM)
http://lsd.ls.fi.upm.es/lsd/lsd.htm

La producción de este material ha sido financiada parcialmente por
Microsoft Research Cambridge (Award MS-2004-393)

Índice

Bibliografía

- H. Attiya, J. Welch. Distributed Computing, 2nd Ed., Wiley-Interscience, 2004. pp. 92-94.
- T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM. March 1996.
- Antonio Fernández Anta. Tolerancia a fallos en sistemas distribuidos. Curso de doctorado. Universidad Rey Juan Carlos. 2006.
- Michael J. Fischer, Nancy A. Lynch and Michael S. Paterson, Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM, April 1985, 32(2):374-382.
- Leslie Lamport, Robert Shostak, Marshall Pease. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems (TOPLAS), n. 3, July 1982. pp. 382 - 401. 1982.
- M. Raynal. A Short Introduction to Failure Detectors for Asynchronous Distributed Systems, ACM SIGACT News Distributed Computing Column 17, March 2005.

Introducción

- El consenso permite a los procesos llegar a un decisión común a partir de valores iniciales y a pesar de fallos.
- El consenso es un problema fundamental en computación distribuida tolerante a fallos.
- Es un denominador común de otros problemas:
  - Radiado atómico, *group membership*, compromiso atómico, elección de líder.
  - Bases de datos replicadas que deben acordar el orden en el que realizan los cambios para mantener la coherencia.
  - Sistemas transaccionales que deben decidir si comprometer o abortar una transacción.

Definición

- Cada proceso p_i propone un valor v_i (*propose(v_i)*) y tarde o temprano cada proceso correcto decide el mismo valor v (*decide(v)*).

  Formalmente:
  - **Terminación**: Todo proceso correcto decide en algún momento.
  - **Acuerdo:** Dos procesos correctos no deciden valores diferentes.
  - **Validez:** El valor decidido debe ser uno de los propuestos.

Definición (cont.)

- **Consenso uniforme**: Dos procesos (correctos o no) nunca deciden valores diferentes.

  Formalmente:
  - **Terminación**: Igual que consenso.
  - **Acuerdo uniforme:** Dos procesos no deciden valores diferentes.
  - **Validez:** Igual que consenso.

Índice

- Introducción.
- Definición.
- ***Modelo con fallos de parada.***
  - ***Consenso síncrono.***
  - Consenso asíncrono.
- Modelo con fallos bizantinos.
  - Consenso síncrono.
  - Consenso asíncrono.

Modelo con fallos de parada

Sistema distribuido con paso de mensajes:

- Conjunto finito de *n* procesos $\Pi=\{p_1, ..., p_n\}$.
- Los procesos envían y reciben mensajes usando los canales.
- Fallos de parada (crash): Un proceso que falla simplemente deja de avanzar (se cae). Si estaba enviando mensajes a otros procesos, puede que algunos se hayan enviado y otros no.

Modelo con fallos de parada. Consenso síncrono

- Modelo de sistema.
- Un ejemplo.
- Cota inferior.

Consenso síncrono. Modelo de sistema

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

- Canales de comunicación bidireccionales que unen cada par de procesos.
- Los canales son fiables (no pierden mensajes).
- Los procesos envían y reciben mensajes con *send(m)* y *receive(m)*
- La mínima velocidad de avance de los procesos y el máximo retardo de los mensajes están acotados y las cotas son conocidas por todos.

Consenso síncrono. Un ejemplo

- *f* es el número máximo de procesos que pueden caerse o fallar (*n-f* son correctos).
- Suponemos *f* conocido (si *f* desconocido, *f=n-1*).
- El algoritmo tarda *f +1* rondas síncronas.
- En cada ronda cada proceso envía a todos los
- demás.
- Eficiencia: tiempo *f +1*, mensajes n(f +1).

Consenso síncrono. Un ejemplo (cont.)

```
Procedure propose(v_p)
   V_p ← {v_p}

// Fase 1: rondas síncronas  1 <= r_p <= f+1
   for r_p=1 to f+1
         send (r_p, V_p) a todos
         wait until [∀q: receive (r_p, V_q) or time-out]
         V_p ← V_p U {V_q | recibido V_q}

// Fase 2:
   decide(mínimo valor de V_p)
```

(Código para el proceso p)

Optimizaciones:

1. No reenviar valores ya enviados: por ser todos los canales fiables
2. No enviar r_p: si un mensaje no llega en tiempo, entonces no llegará nunca

send ($\forall v \in V_p$ | v *was not sent by p*) a todos

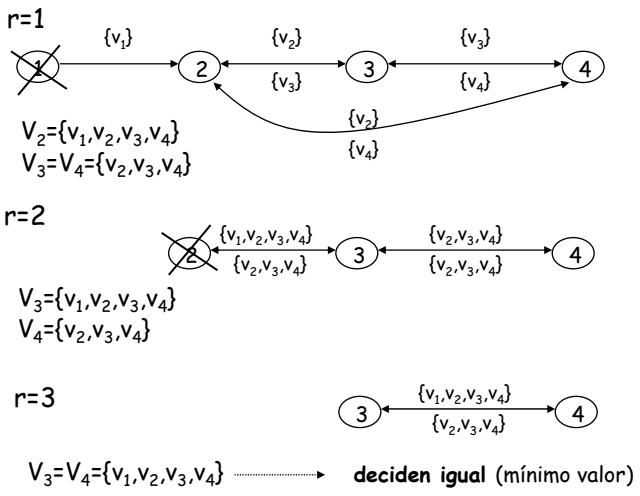
Consenso síncrono. Un ejemplo (cont.)

El algoritmo satisface las propiedades del consenso:

- Terminación: Un proceso correcto decide tras $f+1$ rondas
- Acuerdo: Al final de la ronda $f+1$ todos los procesos correctos tienen idénticos conjuntos V .
- Validez: El conjunto V sólo contiene valores propuestos.

Consenso síncrono. Un ejemplo (cont.)

f=2



Consenso síncrono. Un ejemplo (cont.)

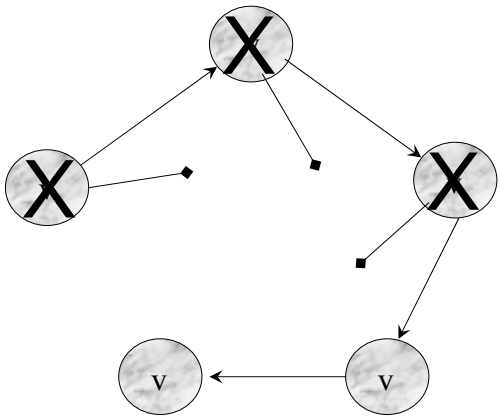
Sea p_i correcto con $x \in V_i$. Supongamos que p_j correcto no tiene x en su V_j:

- Si el primer proceso correcto en incluir x en su V lo hace en la ronda $r \leq f$, en la ronda $(r + 1) \leq (f + 1)$ todos los correctos tienen x en sus V.
- Si no, hay una cadena de procesos fallidos distintos p_{i1}, p_{i2} , ..., p_{if+1} tal que p_{ik} tiene $x \in V$ al comenzar la ronda k y falla al enviar este conjunto (por eso sólo algunos procesos reciben x). Como sólo hay f procesos fallidos esto no es posible.

Consenso síncrono. Cota inferior

- Si $f < n - 1$, no hay ningún algoritmo que resuelva (en el peor caso) consenso en menos de $f+1$ rondas .
  - La idea es que si un proceso decide antes de la ronda f + 1 se puede estar equivocando.
- Si f = n − 1, con f rondas basta.

Consenso síncrono. Cota inferior (cont.)



Índice

- Introducción.
- Definición.
- ***Modelo con fallos de parada.***
  - Consenso síncrono.
  - ***Consenso asíncrono.***
- Modelo con fallos bizantinos.
  - Consenso síncrono.
  - Consenso asíncrono.

Modelo con fallos de parada. Consenso asíncrono

- Imposibilidad.
- Detectores de fallos no fiables.
- Algoritmos con detectores de fallos.
- Cota inferior con detectores de fallos.

Consenso asíncrono. Imposibilidad

Ningún algoritmo determinista resuelve el consenso en un sistema asíncrono en la presencia de uno o más fallos [FLP85].

- Intuición: No es posible distinguir un proceso lento de uno fallido.
- Demostración del resultado para 2 procesos p_1 y p_2.
- Generalización (por simulación) para n procesos.

Consenso asíncrono. Imposibilidad (cont.)

Para 2 procesos p_1 y p_2:

- Ejecución R_1:
  - p_2 falla al inicio y p_1 propone 1.
  - p_1 decide 1 tras un tiempo t_1.
- Ejecución R2:
  - p_1 falla al inicio y p_2 propone 2.
  - p_2 decide 2 tras un tiempo t_2.

- En una ejecución R_3 en que ningún mensaje llega antes de $\max(t_1,t_2)$, ambos deciden diferente y se viola acuerdo.

Consenso asíncrono. Imposibilidad (cont.)

Formas de superar la imposibilidad:

- Algoritmos probabilistas (*randomized algorithms*): Permiten alcanzar el consenso en un tiempo máximo (sin probabilidad 1).
- Sistemas parcialmente síncronos: El tiempo de transmisión de mensajes está acotado, pero no se conoce la cota.
- *Detectores de fallos no fiables* [Chandra & Toueg96]: resuelven consenso en un sistema **asíncrono** incluso con un infinito número de fallos.

Consenso asíncrono. Detectores de fallos

- Definición.
- Propiedades.
- Clasificación.
- Implementación <>P.
- Reducibilidad.

Detectores de fallos. Definición

- Son dispositivos (o servicios) que proporcionan a cada proceso información sobre procesos fallidos.
- La interfaz de p_i es una llamada (D_i) que devuelve una lista de sospechosos.
- Las listas devueltas a distintos procesos pueden ser incoherentes o erróneas.
- Las garantías de un detector se enuncian como propiedades de completitud y precisión.

Detectores de fallos. Propiedades

- El resultado de la llamada (*D*) debe satisfacer dos propiedades complementarias:
- **Completitud** (*completeness*) : garantiza que SÍ vamos a sospechar de todos los procesos fallidos.
- **Precisión** (*accuracy*): garantiza que NO vamos a sospechar de (ningún/algún) proceso correcto.

Detectores de fallos. Propiedades (cont.)

- Completitud:
  - **Fuerte:** Hay un instante de tiempo a partir del cual *todos* los procesos caídos son sospechados permanentemente por *todos* los procesos correctos.
  - **Débil:** Hay un instante de tiempo a partir del cual *todos* los procesos caídos son sospechados permanentemente por *algún* proceso correcto.
- Precisión:
  - **Fuerte:** No se sospecha de un proceso antes de que se caiga.
  - **Débil:** Existe *algún* proceso correcto *q* que no es sospechado por *ningún* proceso.
  - **Tarde o temprano fuerte**: Hay un instante tiempo a partir del cual *ningún* proceso correcto es sospechado por *ningún* proceso correcto.
  - **Tarde o temprano débil:** Hay un instante de tiempo a partir del cual existe *algún* proceso correcto *q* que no es sospechado por *ningún* proceso correcto.

Detectores de fallos. Clasificación

Clases		Precisión			
		Fuerte	Débil	Tarde o temprano fuerte	Tarde o temprano débil
Completitud	Fuerte	*P*	*S*	<>*P*	<>*S*
	Débil	*Q*	*W*	<>*Q*	<>*W*

Detectores de fallos. Implementación <>P

Modelo de sistema **parcialmente síncrono**:

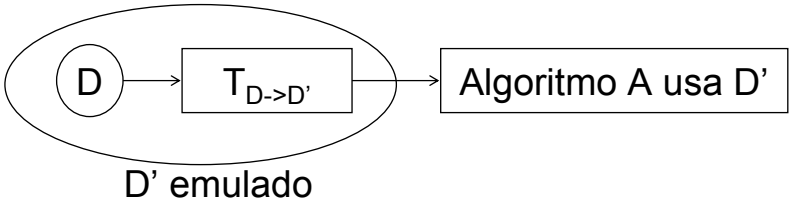
- Canales de comunicación bidireccionales que unen cada par de procesos.
- Los canales son fiables (no pierden mensajes).
- Los procesos envían y reciben mensajes con *send(m)* y *receive(m)*
- La mínima y máxima velocidad de avance de los procesos están acotadas pero las cotas no son conocidas.
- El máximo retardo de los mensajes está acotado pero las cotas no son conocidas.

Detectores de fallos. Implementación <>P (cont.)

- Cada módulo (proceso) manda *periódicamente* un mensaje *latido* (heart-beat) a los demás procesos.
- Si en un módulo *vence el temporizador* sin recibir un latido de otro proceso, se mete a dicho proceso en la lista de sospechosos.
- Si un módulo sospecha de otro y recibe un latido de él, lo quita de la lista de sospechos y *aumenta su temporizador*.
- En un sistema parcialmente síncrono, tarde o temprano el valor del contador será mayor que la cota y no se volverá a sospechar si el proceso es correcto.

Detectores de fallos. Reducibilidad

- Un detector D' se dice que es *más débil* que un detector D ($D \geq D'$) si existe un algoritmo $T_{D->D'}$ que puede transformar D en D'.
- En este caso se dice que D' es *reducible* a D (D proporciona al menos tanta información de fallos como D')
- Reducción trivial: $P \geq Q$, $S \geq W$, $\diamond P \geq \diamond Q$, $\diamond S \geq \diamond W$



Detectores de fallos. Reducibilidad (cont.)

- $T_{D->D'}$ de completitud débil a fuerte:

```
salida_p ← vacio            {salida_p emula D'_p}
cobegin
|| Tarea 1: repeat forever
        {p pregunta a su modulo detector de fallos D_p}
        sospechosos_p ← D_p
        send (p, sospechosos_p) a todos
|| Tarea 2: when receive (q, sospechosos_q) de algún q
        salida_p ← (salida_p U sospechosos_q) - {q} - {p}
coend
```

(Código para el proceso p)

- Con este algoritmo: $Q \geq P$, $W \geq S$, $\Diamond Q \geq \Diamond P$, $\Diamond W \geq \Diamond S$

Detectores de fallos. Reducibilidad (cont.)

- Dos detectores de fallos son *equivalentes* si son reducibles entre sí.
- Por esto:
  - $Q \equiv P$
  - $W \equiv S$
  - $\Diamond Q \equiv \Diamond P$
  - $\Diamond W \equiv \Diamond S$

Consenso asíncrono. Algoritmos con detectores de fallos

- Modelo de sistema.
- Un ejemplo con S.
- Un ejemplo con <>S.

Algoritmos con detectores de fallos. Modelo de sistema

Modelo **asíncrono:**

- Canales de comunicación bidireccionales que unen cada par de procesos.
- Los canales son fiables (no pierden mensajes).
- Los procesos envían y reciben mensajes con *send(m)* y *receive(m)*
- Los procesos del sistema avanzan a cualquier velocidad (> 0) y los mensajes que intercambian pueden sufrir retardos arbitrarios ($\neq\infty$).

Detector de fallos:

- Cada proceso p utiliza D_p para conocer los sospechosos (pueden ser erróneos) que le proporciona su módulo del detector de fallos.

Algoritmos con detectores de fallos. Un ejemplo con S

- Detector S: completitud fuerte y precisión débil.
- S es interesante para consenso porque P ≥ S.
- Tolera hasta *n - 1* procesos fallidos.
- El algoritmo se ejecuta en tres fases: de proposición, de acuerdo y de decisión.
- Por ser W ≡ S, dado un detector W el consenso es resoluble en sistemas asíncronos con $f < n$ (siendo f el número máximo de procesos fallidos)

Algoritmos con detectores de fallos. Un ejemplo con S (cont)

```
Procedure propose(v_p)
    V_p ← [⊥,⊥, ...,⊥]
    V_p[p] ← v_p
    Δ_p ← V_p

// Fase 1: rondas asíncronas r_p, 1 <= r_p <= n – 1
    for r_p = 1 to n - 1
        send (r_p, Δ_p, p) a todos
        wait until [∀q: receive (r_p, Δ_q, q) or q ϵ D_p]
        msgs[r_p] ← {(r_p, Δ_q, q) | recibido (r_p, Δ_q, q) }
        Δ_p ← [⊥,⊥, ...,⊥]
        for k = 1 to n
            if V_p[k] = ⊥ and
            ∃ (r_p, Δ_q, q) ϵ msgs[r_p] con Δ_q[k] ≠ ⊥ then
                V_p[k] ← Δ_q[k]
                Δ_p[k] ← Δ_q[k]

// Fase 2:
  send (V_p) a todos
  wait until [∀q: receive(V_q) or q ϵ D_p]
  lastmsgs_p ← {V_q | recibido V_q}
  for k = 1 to n
      if ∃ V_q ϵ lastmsgs_p con V_q[k] = ⊥
              then
                  V_p[k] ← ⊥

// Fase 3:
 decide(primer componente no-⊥ de V_p)
```

(Código para el proceso p)

Algoritmos con detectores de fallos. Un ejemplo con S (cont)

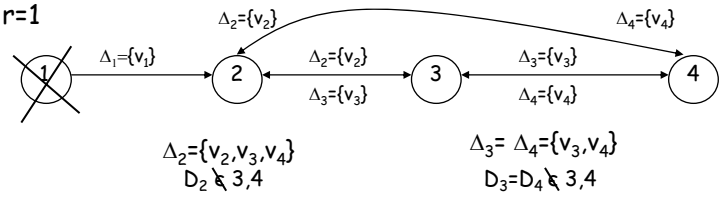
- Hay al menos un proceso no sospechado por nadie (resultado de D)
- Sea p uno de dichos procesos:
  - La información de p siempre llega a todos los procesos
  - En cada ronda p dice lo nuevo que sabe a todos, pero puede aprender algo nuevo
  - Llega un momento en que p ya no aprende nada nuevo (después de n-1 rondas)
  - En la última ronda (fase 2), p mandará y tendrá un vector con la mínima información
- Todos los procesos se deciden por el vector de mínima información que es el mismo.
- Todos los procesos aplican una función determinista sobre el vector

Algoritmos con detectores de fallos. Un ejemplo con S (cont)

f=2

Consideremos que el detector nunca sospecha de 3 y 4: $D_i \not\ni 3,4 \quad \forall i$

Fase 1: durante **f-1** rondas un proceso aprende de posibles caídas



Si suponemos en r=1 que: $D_2 \not\ni 1$ y $D_3, D_4 \in 2$

$\Delta_2=\{\mathbf{v_1}, v_2, v_3, v_4\}$

$\Delta_3 = \Delta_4 = \{v_3, v_4\}$

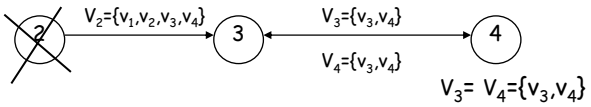
Algoritmos con detectores de fallos. Un ejemplo con S (cont)

f=2

Consideremos que el detector nunca sospecha de 3 y 4: $D_i \not\in 3,4 \quad \forall i$

Fase 2:

r=2



$V_3 = V_4 = \{v_3,v_4\}$

$D_3 = D_4 \not\in 3,4$

Si suponemos en r=2 que: $D_3 \not\in 2$ y $D_4 \in 2$

$V_3=\{v_1,v_2,v_3,v_4\}$ $V_4=\{v_3,v_4\}$

Fase 3:

r=2 $V_3=\{v_1,v_2,v_3,v_4\}$ ⟶ **deciden v_3** (primer valor intersección)

$V_4=\{\quad v_3,v_4\}$

Algoritmos con detectores de fallos. Un ejemplo con <>S

- Detector <>S: completitud fuerte y tarde o temprano precisión débil.
- <>S es interesante para consenso porque <>P $\geq$ <>S.
- Tolera hasta n/2 - 1 procesos fallidos.
- Paradigma del coordinador rotante. En cada ronda cambia el coordinador.
- Cada ronda pasa por 3 fases: proposición, asentimiento y decisión.
- En cada ronda los mensajes van desde el coordinador a todos (propuesta) o salen de todos (asentimiento).
- El algoritmo pasa por 3 épocas: valor no seleccionado, valor seleccionado y valor decidido.

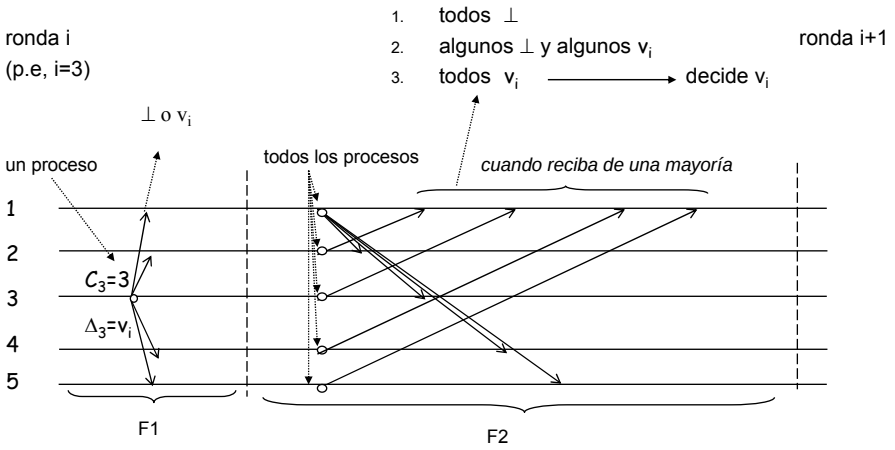
Algoritmos con detectores de fallos. Un ejemplo con <>S (cont.)

Initially:
  start T1 y T2
Task T1:
  $r_p \leftarrow 0$
  $\Delta_p \leftarrow$ propose(v_p)
while *true* **do**
  $r_p \leftarrow r_p + 1$ *// infinitas rondas asíncronas*
  $c_p \leftarrow (r_p \bmod n)+1$
// Fase 1: *del proceso coordinador a todos*
  if ($p = c_p$) **then** send (FASE1,r_p, Δ_p) a todos
  wait until [(receive (FASE1,r_p, Δ) de c_p)
      or $c_p \in D_p$]
  if (recibido (FASE1,r_p, Δ) de c_p) **then**
    $aux_p \leftarrow \Delta$
  else
    $aux_p \leftarrow \perp$

// Fase 2: *de todos a todos*
  send (FASE2,r_p, aux_p) a todos
  wait until [receive (FASE2,r_p, aux_*) de
      la mayoria de procesos]
  let rec_p **be** el conjunto de aux_*
      recibidos en la línea anterior
  case $rec_p = \{\Delta\}$ **then**
    $\Delta_p \leftarrow \Delta$
    send (DECISION, Δ_p) a todos
    stop T1
  case $rec_p = \{\Delta, \perp\}$ **then**
    $\Delta_p \leftarrow \Delta$
  case $rec_p = \{\perp\}$ **then**
    skip
endwhile

Task T2: **when** recibido (DECISION, Δ_{final}):
  send (DECISION, Δ_{final}) a todos; decide(Δ_{final})

Algoritmos con detectores de fallos. Un ejemplo con <>S



Algoritmos con detectores de fallos. Un ejemplo con <>S

- Hay tres épocas:
  - Primera época: el valor estimado no está fijado ya que los mensajes del coordinador no llegan a una mayoría
  - Segunda epoca: empieza cuando una mayoría de procesos recibe el valor del coordinador. Queda fijado el valor.
  - Tercera epoca: empieza después de fijado el valor (2ª época). Se puede radiar fiablemente el valor decidido.
- El algoritmo puede terminar en la primera ronda si las comunicaciones "van bien".
- Como muy tarde, el algoritmo terminará cuando el coordinador sea el proceso correcto no sospechado por nadie (esto ocurre tarde o temprano).

Consenso asíncrono. Cota inferior con detectores de fallos

Ningún algoritmo usando <>P puede implementar consenso en sistemas asíncronos donde $f \geq n/2$

2 conjuntos Π_1 y Π_2 con $|n/2|$ cada uno. Todos los proceso de Π_1 proponen 1 y todos los proceso de Π_2 proponen 2. p_1 está en Π_1 y p_2 está en Π_2

- Ejecución R_1:
  - Todos los procesos de Π_2 fallan al inicio y todos los procesos de Π_1 proponen 1.
  - p_1 decide 1 tras un tiempo t_1.
- Ejecución R_2:
  - Todos los procesos de Π_1 fallan al inicio y todos los procesos de Π_2
  - p_2 decide 2 tras un tiempo t_2.

- En una ejecución R_3 en que ningún mensaje llega antes de $\max(t_1,t_2)$, ambos deciden diferente y se viola acuerdo.

Índice

- Introducción.
- Definición.
- Modelo con fallos de parada.
  - Consenso síncrono.
  - Consenso asíncrono.
- ***Modelo con fallos bizantinos.***
  - ***Consenso síncrono.***
  - Consenso asíncrono.

Modelo con fallos bizantinos

El problema de los generales bizantinos [Lamport]:

- Tres o más generales tienen que decidir atacar o no.
- Uno de los generales (comandante) y da la orden al resto de los generales (lugartenientes).
- Éste puede engañar, decir una cosa a un general y otra a otro. Si el general es correcto, todos los procesos deciden el valor propuesto por éste.
- Un general puede decir a otro que el general le ordenó atacar y a otro decirle que no.
- A diferencia del consenso, un proceso propone un valor y los demás deben decidir si están de acuerdo o no en ese valor.

Modelo con fallos bizantinos (cont.)

Sistema distribuido con paso de mensajes:

- Conjunto finito de *n* procesos $\Pi=\{p_1, ..., p_n\}$.
- Los procesos envían y reciben mensajes usando los canales.
- Fallos bizantinos: El comportamiento de un proceso fallido no está limitado. Su comportamiento puede ser malicioso.

Modelo con fallos bizantinos. Consenso síncrono

- Modelo de sistema.
- Cota inferior.

Consenso síncrono. Modelo de sistema

En un sistema distribuido **síncrono** con paso de mensajes:

- Canales de comunicación bidireccionales que unen cada par de procesos.
- Los canales son fiables (no pierden mensajes).
- Los procesos envían y reciben mensajes con *send(m)* y *receive(m)*
- La mínima velocidad de avance de los procesos y el máximo retardo de los mensajes están acotados y las cotas son conocidas por todos.

- Los procesos tienen fallos arbitrarios: pueden enviar mensajes con cualquier valor, o no enviar mensajes.

Consenso síncrono. Cota inferior.

No es posible resolver el problema en un sistema con $n \leq 3f$ [Lamport82].

- Supongamos que existe un algoritmo A. Lo usamos para resolver consenso con 3 procesos.
- 3 procesos simulan la ejecución de A en n procesos virtuales: cada uno no más de f.
- Si falla uno de los 3, fallan como mucho f procesos virtuales.
- Los otros dos procesos deben decidir correctamente: contradicción.

Índice

- Introducción.
- Definición.
- Modelo con fallos de parada.
  - Consenso síncrono.
  - Consenso asíncrono.
- ***Modelo con fallos bizantinos.***
  - Consenso síncrono.
  - ***Consenso asíncrono.***

Modelo con fallos bizantinos. Consenso asíncrono

- Modelo de sistema.
- Un ejemplo.

Consenso asíncrono. Modelo de sistema

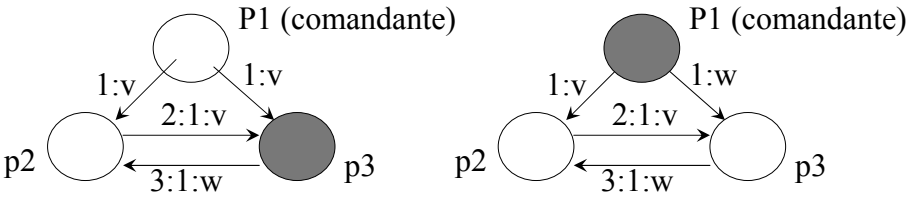
En un sistema distribuido **asíncrono** con paso de mensajes:

- Canales de comunicación bidireccionales que unen cada par de procesos.
- Los canales son fiables (no pierden mensajes).
- Los procesos envían y reciben mensajes con *send(m)* y *receive(m)*
- Los procesos del sistema avanzan a cualquier velocidad (> 0) y los mensajes que intercambian pueden sufrir retardos arbitarios ($\neq\infty$).

- Los procesos tienen fallos arbitrarios: pueden enviar mensajes con cualquier valor, pueden no enviar mensajes, pueden enviar mensajes en cualquier momento.

Consenso asíncrono. Un ejemplo

Modelo bizantino



- Si hay firmas digitales, puede haber acuerdo con tres procesos y un fallo.

Agradecimientos

- A Antonio Fernández Anta y a Sergio Arévalo, de la Universidad Rey Juan Carlos, por permitir utilizar parte de su trabajo para la realización de esta documentación.