

# Sistemas Distribuidos

## Comunicación por mensajes

Sergio Yovine

Departamento de Computación, FCEyN,  
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2017

## (2) Problemas

- Orden de ocurrencia de los eventos
- Exclusión mutua
- Consenso

### (3) Exclusión mutua: comunicación por mensajes

#### Requerimiento

- No se pierden mensajes
- Ningún proceso falla

#### Algoritmos

- Lamport (1978)
  - Orden total
- *Token passing*
  - Fiber Distributed Data Interface (FDDI)
  - Time-Division Multiple-Access (TDMA)
    - Timed-Triggered Architecture (TTA)

#### Propiedades

- EXCL
- G-PROG
- Justicia (*fairness*)

## (4) Orden de ocurrencia de los eventos: Lamport (1978)

### Relojes

- Un **reloj** es una función que asigna un *valor* a cada evento
- Ese valor *representa* el momento en que el evento  $e$  ocurrió
- Cada proceso (o nodo)  $i$  tiene un reloj  $C_i$ .
- El reloj **global**  $C$  es tal que  $C(e) = C_i(e)$  si  $e$  ocurre en  $i$ .

### Eventos

- $a \rightarrow b$  si  $a$  **ocurre antes que**  $b$ .
- $\rightarrow$  es un *orden parcial no reflexivo*.
- Si  $a \rightarrow b$  y  $b \rightarrow c$ , entonces  $a \rightarrow c$ .
- Si  $\neg(a \rightarrow b \vee b \rightarrow a)$ , entonces  $a$  y  $b$  son **concurrentes**.

## (5) Orden de ocurrencia de los eventos: Lamport (1978)

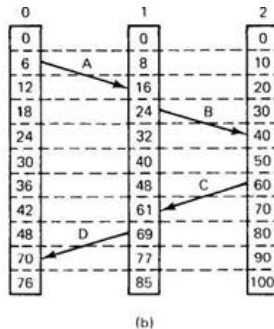
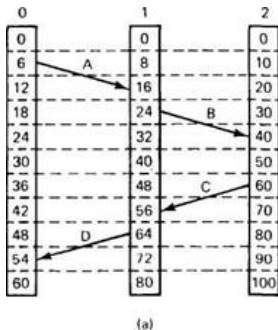
Propiedad a satisfacer:

- Si  $a$  y  $b$  ocurren en  $i$  y  $a \rightarrow b$ , entonces  $C_i(a) < C_i(b)$ .
- Si  $e = snd_i(m)$  y  $r = rcv_j(m)$ , entonces  $C_i(e) < C_j(r)$ .

## (6) Orden de ocurrencia de los eventos: Lamport (1978)

Algoritmo:

- $i$  incrementa  $C_i$  entre todo par de eventos consecutivos.
- $i$  envía:  $e = snd_i(m, C_i(e))$ .
- $j$  recibe:  $r = rcv_j(m, t)$ ,  $C_j(m) = t' > t$ .



## (7) Exclusión mutua: Lamport (1978)

### Orden total

- $C(a) < C(b)$  sii  $C_i(a) < C_j(b)$  o  $C_i(a) = C_j(b) \wedge i < j$ .

### Acciones proactivas

- $try_i$ :  $i$  manda  $(i, req)$  a todos y lo guarda
- $exit_i$ :  $i$  borra  $(i, req)$  y envía  $(i, rel)$  a todos
- $crit_i$ :
  - hay un mensaje  $m = (i, req)$  en la cola de pedidos de  $i$
  - $C(m) < C(m')$  para **todo**  $m' = (i', req)$  en la cola
  - $i$  recibió **todos** los mensajes de  $ack$  posteriores a  $m$

### Acciones reactivas (invisibles)

- en  $T_i$ ,  $i$  recibe  $(j, ack)$ : lo guarda
- $i$  recibe  $(j, req)$ : lo guarda y manda un  $(i, ack)$  a  $j$
- $i$  recibe  $(j, rel)$ : borra  $(j, req)$

## (8) Consenso

Todos los procesos tienen que estar de **acuerdo**

Un *problema de consenso* tiene la siguiente forma general:

**Valores**  $V$  es un conjunto de valores a decidir, e.g,  $\{0, 1\}$

**Inicio** Todo proceso  $i$  empieza con  $in(i) \in V$

**Acuerdo** Para todo  $i \neq j$ ,  $decide(i) = decide(j)$

**Validez** Existe  $i$ ,  $in(i) = decide(i)$

**Terminación** Todo  $i$  decide en un número finito de transiciones

Valores, acuerdo, validez y terminación cambian según el problema particular de consenso

Si no hay fallas, el problema tiene solución

Veamos algunos ejemplos



## (9) Consenso: Elección de líder (sin fallas)

- En un anillo sin fallas con comunicación sincrónica
- Le Lann, Chang y Roberts (N. Lynch, Cap. 3 y Cap. 15.1)
- Algoritmo
  - Valores:  $V = \{pid(i) \mid 1, \dots, N\}$
  - Inicio: Todo proceso  $i$  envía su  $pid\ pid(i)$
  - Cuando  $i$  recibe  $p$ :
    - Si  $pid(i) < p$ ,  $i$  propaga  $p$
    - Si  $pid(i) > p$ ,  $i$  descarta  $p$
    - Si  $pid(i) = p$ ,  $i$  se declara líder,  $decide(i) = p$  y envía  $stop(p)$
  - Cuando  $i$  recibe  $stop(p)$ ,  $decide(i) = p$
- Tiempo
  - Sin fase de  $stop\ \mathcal{O}(n)$
  - Con fase de  $stop\ \mathcal{O}(2 \cdot n)$
- Comunicación
  - $\mathcal{O}(n^2)$
  - Cota inferior  $\Omega(n \log n)$ . Algoritmo de Hirschberg y Sinclair.

## (10) Consenso: Commit en una BD distribuida (sin fallas)

Descripción (**COMMIT**)

Valores  $V = \{0 \text{ (abort)}, 1 \text{ (commit)}\}$

Acuerdo  $\forall i \neq j. \text{decide}(i) = \text{decide}(j)$

Validez

$$\textcircled{1} \exists i. \text{in}(i) = 0 \implies \forall i. \text{decide}(i) = 0$$

$$\textcircled{2} \forall i. \text{in}(i) = 1 \implies \forall i. \text{decide}(i) = 1$$

Terminación Todo  $i$  decide en un número finito de transiciones

# (11) Consenso: Commit en una BD distribuida (sin fallas)

## Two-phase commit (2PC)

Se elige un proceso distintivo, por ejemplo, 1

- Fase 1

- 1  $\forall i \neq 1$ :  $i$  envía  $in(i)$  a 1. Si  $in(i) = 0$ ,  $decide(i) = 0$ .
- 2  $i = 1$ : Si recibe todos 1,  $decide(i) = in(i)$ , si no,  $decide(i) = 0$ .

- Fase 2

- 1  $i = 1$ : Envía  $decide(i)$  a todos.
- 2  $\forall i \neq 1$ : Si  $i$  no decidió,  $decide(i)$  es el valor recibido de 1.

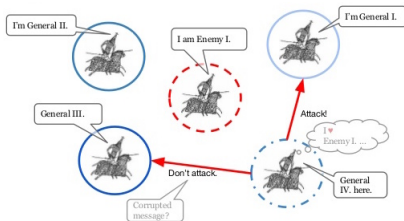
## (12) Consenso con fallas

¿Qué pasa si hay fallas?

- Problema del *ataque coordinado* o de los Generales Bizantinos
- *commit* (attack) o *abort* (don't attack) en una transacción

Hay tres tipos de fallas

- Falla la comunicación
- Los procesos dejan de funcionar
- Los procesos no son confiables (falla bizantina)



## (13) Consenso: falla la comunicación

### Descripción

**Valores**  $V = \{0, 1\}$

**Inicio** Todo proceso  $i$  empieza con  $in(i) \in V$

**Acuerdo** Para todo  $i \neq j$ ,  $decide(i) = decide(j)$

**Validez** Existe  $i$ ,  $in(i) = decide(i)$

**Terminación** Todo  $i$  decide en un número finito de transiciones

### Teorema

No existe ningún algoritmo para resolver consenso

## (14) Consenso: los procesos dejan de funcionar

### Descripción

**Valores**  $V = \{0, 1\}$

**Inicio** Todos proceso  $i$  empieza con  $in(i) \in V$

**Acuerdo**  $\nexists i \neq j. decide(i) \neq decide(j)$

**Validez** Existe  $i, in(i) = decide(i)$

**Terminación** Todo  $i$  que *no falla* decide en un número finito de transiciones

### Teorema

Si fallan a lo sumo  $f < n$  procesos, entonces se puede resolver consenso con  $\mathcal{O}((f + 1) \cdot n^2)$  mensajes

## (15) Consenso: Commit en una BD distribuida

### Descripción (**COMMIT**)

**Valores**  $V = \{0 \text{ (abort)}, 1 \text{ (commit)}\}$

**Acuerdo**  $\nexists i \neq j. \text{decide}(i) \neq \text{decide}(j)$

**Validez** ①  $\exists i. \text{in}(i) = 0 \implies \nexists i. \text{decide}(i) = 1$

②  $\forall i. \text{in}(i) = 1 \wedge \text{no fallas} \implies \nexists i. \text{decide}(i) = 0$

**Term. débil** Si no hay fallas, todo proceso decide

**Term. fuerte** Todo proceso que no falla decide

### Teorema

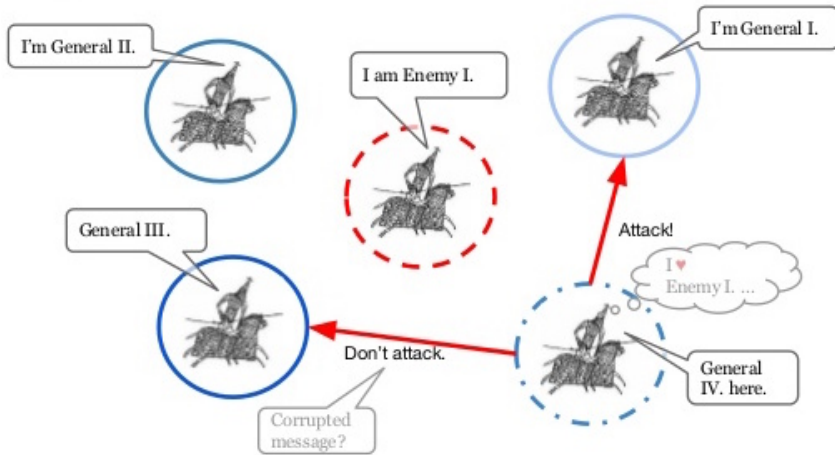
Two-phase commit resuelve **COMMIT** con terminación débil

### Pero

- Two-phase commit no satisface *terminación fuerte*
- Solución: three-phase commit (N. Lynch, Cap. 7.2 y 7.3)

## (16) Consenso: los procesos no son confiables

Falla bizantina (Lamport, 1982)





## (17) Consenso: los procesos no son confiables

### Descripción

**Valores**  $V = \{0, 1\}$

**Inicio** Todos proceso  $i$  empieza con  $in(i) \in V$

**Acuerdo**  $\forall i \neq j$ , que *no fallan*,  $decide(i) = decide(j) \in V$

**Validez** Si  $\forall i$ , que *no falla*,  $in(i) = v$ , entonces  $\nexists j$ , que *no falla*, tal que  $decide(j) \neq v$

**Terminación** Todo  $i$  que *no falla* decide en un número finito de transiciones

### Teorema

Se puede resolver consenso bizantino para  $n$  procesos y  $f$  fallas si y sólo si  $n > 3 \cdot f$  y la *conectividad* es mayor que  $2 \cdot f$

Conectividad:  $\text{conn}(G) = \text{mínimo número de nodos } N \text{ tal que } G \setminus N \text{ no es conexo o es trivial}$

## (18) Consenso: Otros tipos de acuerdo y aplicaciones

### Acuerdos

- $k$ -agreement (o  $k$ -set agreement)

$$\text{decide}(i) \in W, \text{ tal que } |W| = k$$

- Aproximado

$$\forall i \neq j. |\text{decide}(i) - \text{decide}(j)| \leq \epsilon$$

- Probabilístico

$$\Pr[\exists i \neq j. \text{decide}(i) \neq \text{decide}(j)] < \epsilon$$

### Aplicaciones

- Sincronización de relojes (NTP, RFC 5905 y anteriores)
- Tolerancia a fallas en sistemas críticos

## (19) Bibliografía extra

- L. Lamport. Time, clocks, and the ordering of events in a distributed system. CACM 21:7 1978.<http://goo.gl/ENh2f7>
- L. Lamport, R. Shostak, M. Pease. The Bizantine Generals problem. ACM TOPLAS 4:3, 1982.<http://goo.gl/DY0Qis>
- Hermann Kopetz, Günther Bauer: The time-triggered architecture. Proceedings of the IEEE 91(1): 112-126 (2003). <http://goo.gl/RPqfas>
- R. Jain. FDDI Handbook. Addison Wesley, 1994. <http://goo.gl/YZ2Hy1>