Sistemas distribuidos
Comunicación por mensajes

Sergio Yovine

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

Sistemas Operativos, segundo cuatrimestre de 2015

(2) Problemas

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

(3) 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*.

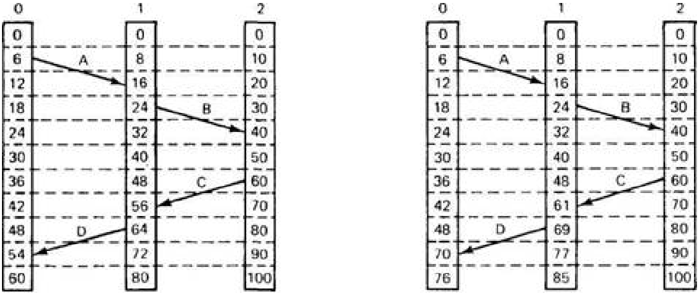
(4) Orden de ocurrencia de los eventos: Lamport (1978)

Propiedad a satisfacer:

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

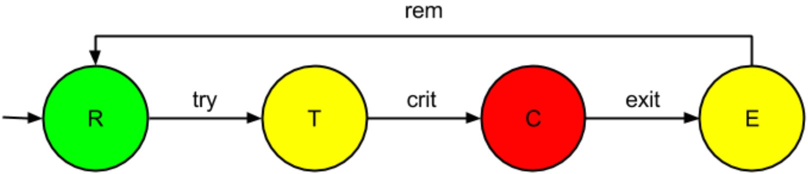
Algoritmo:

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



(a) (b)

- N. Lynch, Distributed Algorithms, 1996 (Cap. 10)



- Estado: $\sigma : [1 \ldots n] \mapsto \{R, T, C, E\}$
- Transición: $\sigma \xrightarrow{\ell} \sigma'$, $\ell \in \{try, crit, exit, rem\}$
- Ejecución: $\tau = \tau_0 \xrightarrow{\ell} \tau_1 \ldots$
- Sección crítica: $CRIT \equiv \{i \mid \sigma(i) = C\}$
- Sección pre-crítica: $TRY \equiv \{i \mid \sigma(i) = T\}$

(6) Exclusión mutua: propiedades

Exclusión mutua (EXCL)

Para toda ejecución τ y estado τ_k, no puede haber más de un proceso i tal que $\tau_k(i) = C$.

$$\Box \, \# CRIT \leq 1$$

Progreso (PROG)
(*lock-free*)

Para toda ejecución τ y estado τ_k,
si en τ_k hay un proceso i en T y ningún proceso en C
entonces $\exists j > k$, t. q. en el estado τ_j algún proceso i' está en C.

$$\Box\,(\#TRY \leq 1 \wedge \#CRIT = 0) \implies \Diamond \#CRIT > 0$$

Progreso global absoluto (WAIT-FREE)

Para toda ejecución τ, estado τ_k y todo proceso i,
si $\tau_k(i) = T$
entonces $\exists j > k$, tal que $\tau_j(i) = C$.

$$IN(i) \quad \equiv \quad i \in TRY \implies \Diamond\, i \in CRIT$$

$$\forall i. \Box\, IN(i)$$

Progreso global dependiente (G-PROG)

(*deadlock-, lockout-, o starvation-free*)

Para toda ejecución τ,

si para todo estado τ_k y proceso i tal que $\tau_k(i) = C$,
$\exists j > k$ tal que $\tau_j(i) = R$

entonces para todo estado $\tau_{k'}$ y todo proceso i',
si $\tau_{k'}(i') = T$, entonces $\exists j' > k'$, tal que $\tau_{j'}(i') = C$.

$$OUT(i) \quad \equiv \quad i \in CRIT \implies \Diamond\, i \in REM$$

$$\forall i. \Box\, OUT(i) \quad \implies \quad \forall i. \Box\, IN(i)$$

(10) Exclusión mutua: comunicación por mensajes

Requerimiento

- No se pierden mensajes
- Ningún proceso falla

Algoritmos

- Lamport (1978)
  - Orden total (ordenando eventos concurrentes por el *pid*).
- *Token passing*
  - Fiber Distributed Data Interface (FDDI)
  - Time-Division Multiple-Access (TDMA)
    - Timed-Triggered Architecture (TTA)

Propiedades

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

(11) Exclusión mutua: Lamport (1978)

Acciones proactivas

- try_i: i manda (i, req) a todos y lo guarda
- $exit_i$: i borra todos los mensajes (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 todos los mensajes (j, req)

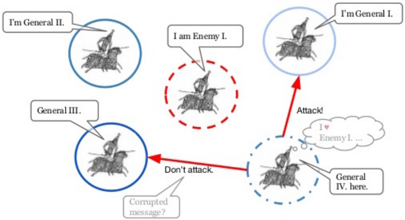
Todos los procesos tienen que estar de *acuerdo*
Si no hay fallas, el problema tiene solución

¿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** Todos proceso i empieza con $in(i) \in V$
- **Acuerdo** Para todo $i \neq j$, $decide(i) = decide(j)$
- **Validez**
  1. Para todo i, si $in(i) = 0$ entonces $decide(i) = 0$
  2. Para todo i, si $in(i) = 1$ y ningún mensaje se pierde, entonces $decide(i) = 1$
- **Terminación** Todo i decide en un número finito de transiciones (*wait-free*)

Teorema

No existe ningún algoritmo para resolver consenso

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 Si $\forall i.\ in(i) = v$, entonces $\nexists j.\ decide(j) \neq v$

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: 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 $out(j) \neq v$
Terminación	Todo i que *no falla* decide en un número finito de transiciones

Teorema

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

Conectividad: conn(G) = mínimo número de nodos N t.q. $G \setminus N$ no es conexo o es trivial

(16) Consenso: Elección de líder

- En un anillo sin fallas con comunicación sincrónica
- Le Lann, Chang y Roberts (N. Lynch, Cap. 3 y Cap. 15.1)
  - Para todo $i \neq j$, $pid(i) \neq pid(j)$
  - 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* (y envía *stop*)
- 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.

Descripción (**COMMIT**)

Valores	$V = \{0(abort), 1(commit)\}$
Acuerdo	$\nexists i \neq j.\ decide(i) \neq decide(j)$
Validez	1. $\exists i.\ in(i) = 0 \implies \nexists i.\ decide(i) = 1$ 2. $\forall i.\ in(i) = 1 \wedge \text{no fallas} \implies \nexists i.\ decide(i) = 0$
Term. débil	Si no hay fallas, todo proceso decide
Term. fuerte	Todo proceso que no falla decide

Two-phase commit

- 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.

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)

(19) Consenso: Otros tipos de acuerdo y applicaciones

Acuerdos

- *k*-agreement (o *k*-set agreement)

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

- Aproximado

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

- Probabilístico

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

Aplicaciones

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

(20) 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/YZ2Hyl`