Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida

Tarea 8

Johann Ramón Gordillo Guzmán

418046090

Tarea presentada como parte del curso de **Computación Distribuida** impartido por la profesora **M.C Karla Rocío Vargas Godoy**.

08 de Diciembre del 2020

Link al código fuente: https://github.com/JohannGordillo/

Actividades

1. (2.5 ptos) Demuestra que el siguiente algoritmo implementa un detector de la clase $\Diamond P$.

Algorithm 1 Code for process p

Constants
  $neighbors$ ▷ list of neighbors of p
  T ▷ integer, time between successive heartbeats
  n ▷ integer, number of processes in the network
Variables
  $clock()$ ▷ local clock of process p
  $lastHB = [0, \dots, 0]$ ▷ array of last time p received a pair labeled with a process
  $timeout = [T, \dots, T]$ ▷ array of estimated timeouts for all processes
  $suspect = [false, \dots, false]$ ▷ array of booleans for suspecting processes

  every T units of time
  for each $q \in neighbors$ **do**
    send($\langle HB, p\rangle$)
  end for

  upon the receiving of $\langle HB, q\rangle$ from a neighbor q
  begin:
    if ($suspect[q] = true$) **then**
      $timeout[q] = timeout[q] + 1$
      $suspect[q] = false$
    end if
    $lastHB[q] = clock()$
  end

  when $timeout[q] == clock() - lastHB[q]$ ▷ when the timer for a message of $q \in \Pi$ expires, p starts suspecting q
  begin:
    $suspect[q] = true$
  end

Demostración.

Para probar que el algoritmo anterior implementa un detector de la clase $\Diamond P$ tenemos que probar las dos propiedades que lo caracterizan:

- **Eventual Strong Accuracy.**

  Esta propiedad nos dice que hay un tiempo después del cual todos los procesos correctos nunca están bajo sospecha por ningún proceso correcto.

  La demostración de esta propiedad la vimos en clase.

  $\therefore$ Se cumple Eventual Strong Accuracy.

- **Strong Completeness.**

  Esta propiedad nos dice que todo proceso fallido es eventualmente sospechado por todo proceso correcto.

Sean $p_i, p_j \in \Pi$ tales que p_i es un proceso correcto y que p_j falla en un momento determinado de la ejecución del algoritmo.

En el momento en que p_j falla, deja de enviar mensajes HB a sus vecinos, incluyendo a p_i, por ser el sistema distribuido una red completa.

Dado que p_i ya no recibe mensajes de p_j, llegará un momento en el que $timeout_i[j] = clock_i() - lastHB[j]$, por lo que en ese momento el proceso p_i hará $suspect_i[j] = True$.

Ahora, dado que p_j ya no enviará mensajes a p_i, no se incrementará $timeout_i[j]$ y $suspect_i[j]$ será falso hasta que termine la ejecución del algoritmo.

$\therefore$ Se cumple Strong Completeness.

Otra manera más breve de ver la demostración es considerando que cuando un proceso envía un mensaje a otro, éste puede estar en la lista de sospechosos, sin embargo llegará un momento en que el sistema se estabilice, y en este momento, cuando un proceso considerado sospechoso dé señales de vida, dejará de ser considerado sospechoso permanentemente, por lo que todo proceso fallido será considerado como sospechoso permanentemente, además de que se dejará de sospechar de los procesos correctos.

$\therefore$ El algoritmo implementa un detector de la clase $\Diamond P$. ■

2. (2.5 ptos) El problema del k-acuerdo es un uno en el que queremos que cada uno de los n procesos decida algún valor con la propiedad de que el conjunto de decisión tenga a lo más k elementos diferentes. Se sabe que el problema del k-acuerdo no puede ser resuelto deterministicamente en sistemas asíncronos con k o más fallas. Supongamos que estamos trabajando en un sistema asíncrono y que tiene disponible el detector $\Diamond S$. ¿Es posible resolver el problema del k-acuerdo con f fallas cuando $k \leq f < \frac{n}{2}$ usando $\Diamond S$?

Respuesta.

Consideremos un sistema asíncrono con el detector de fallas $\Diamond S$ disponible. Recordemos que este detector tiene las siguientes propiedades:

- **Strong Completeness.**

  Esta propiedad nos dice que todo proceso fallido es eventualmente sospechado por todo proceso correcto.

- **Eventual Weak Accuracy.**

  Esta propiedad nos dice que hay un tiempo después del cual algún proceso correcto nunca está bajo sospecha por ningún proceso correcto.

Notemos además que para el caso en el que $k = 1$ hay que suponer una mayoría de procesos correctos, es decir $f < \frac{n}{2}$, pues el detector es erratico en un inicio.

Para $k > 1$ también hay que seguir pidiendo lo mismo, pues esta condición es necesaria para que ningún proceso se quede colgado y así asegurar la terminación del algoritmo, ya que las propiedades del detector de fallas son eventuales y éste es erratico al inicio de la ejecución. Por lo tanto sí es posible resolver el problema del k-acuerdo con f fallas cuando $k \leq f < \frac{n}{2}$.

3. (2.5 ptos) Si tenemos disponible el detector $\Diamond P$, ¿es necesario seguir suponiendo que $f < \frac{n}{2}$ para resolver el consenso usando $\Diamond P$?

Respuesta.

Sí. Recordemos que los detectores de la clase $\Diamond P$ cumplen dos propiedades:

- **Strong Completeness.**

  Esta propiedad nos dice que todo proceso fallido es eventualmente sospechado por todo proceso correcto.

- **Eventual Strong Accuracy.**

  Esta propiedad nos dice que hay un tiempo después del cual todos los procesos correctos nunca están bajo sospecha por ningún proceso correcto.

Sin embargo, dado que las propiedades son eventuales, no se cumplirá desde el inicio que un proceso fallido será sospechado por todos los procesos correctos ni que un proceso correcto no será sospechado por otro proceso correcto. Entonces el problema está en que el detector de fallos puede mentirnos en un inicio por un intervalo largo del protocolo y el consenso necesita terminar eventualmente a pesar de estas mentiras.

Cabe añadir que la demostración formal de este resultado está en el Teorema 6.3.1 de [1]. La demostración se realiza por contradicción. Se supone que la mitad de los procesos toman entrada 0 y la otra mitad toma entrada 1, y cada mitad ejecuta el algoritmo de manera independiente con $\Diamond P$ sospechando de la otra mitad hasta que los procesos en ambas particiones decidan. Los procesos en la primera partición deciden 0 y los procesos en la segunda partición deciden 1. Posteriormente se propone una tercera ejecución en la que de nueva cuenta todos los procesos de la primera partición proponen 0 y todos los procesos de la segunda partición proponen 1. Eventualmente ningún proceso sospecha de otro y llega un momento en el que los procesos de la primera partición deciden 0 y los procesos de la segunda partición proponen 1, pero esto es una violación al acuerdo necesario para resolver el consenso.

Por lo tanto, es necesario seguir suponiendo que $f < \frac{n}{2}$ para resolver el consenso usando $\Diamond P$.

4. (2.5 ptos) Considera el siguiente detector de fallas:

Definición 1. *El detector de fallas* Ω *satisface las siguientes propiedades:*

- Validez: *Cada invocación de* Ω *regresa el nombre de un proceso.*

- Liderazgo eventual: $\forall t' \geq t$ *y para algún proceso correcto* p*, cada invocación de* Ω *por cada proceso correcto, regresa* p*.*

¿Podemos resolver el consenso utilizando el detector de fallas Ω?

Respuesta.

Sí. Más aún, cualquier detector de fallas más débil que Ω no puede resolver el consenso en sistemas asíncronos con fallas. Se le conoce como el líder eventual y es una cota inferior computacional para el consenso con una mayoría de procesos correctos.

Podemos considerar el algoritmo descrito en la figura 17.4 de [3] donde se utiliza el detector de fallas Ω para resolver el consenso con una mayoría de procesos correctos:

```
operation propose (v_i) is
est1_i ← v_i; r_i ← 0;
while (true) do
    begin asynchronous round
    r_i ← r_i + 1;
    % Phase 1 : select a value with the help of Ω %
      my_leader_i ← leader_i; % read the local output of Ω %
      broadcast PHASE1 (r_i, est1_i, my_leader_i);
      wait (   (PHASE1 (r_i, −, −) received from (n − t) processes)
            ∧ (PHASE1 (r_i, −, −) received from p_{my_leader_i} ∨ my_leader_i ≠ leader_i));
      if ((∃ℓ: PHASE1 (r_i, −, ℓ) received from > n/2 processes) ∧ ((r_i, v, −) received from p_ℓ))
                             then est2_i ← v else est2_i ← ⊥ end if;
    % Here, we have ((est2_i ≠ ⊥) ∧ (est2_j ≠ ⊥)) ⇒ (est2_i = est2_j = v) %
    % Phase 2 : try to decide a value from the est2 values %
      broadcast PHASE2 (r_i, est2_i);
      wait (PHASE2 (r_i, −) received from (n − t) processes);
      let rec_i = {est2 | PHASE2 (r_i, est2) has been received};
      case (rec_i = {v})       then broadcast DECIDE(v); return(v)
           (rec_i = {v, ⊥})    then est1_i ← v
           (rec_i = {⊥})       then skip
      end case
    end asynchronous round
end while.

when DECIDE(v) is received do broadcast DECIDE(v); return(v).
```

El algoritmo consta de dos fases. Durante la primera fase los procesos se ayudan del detector de fallos Ω para seleccionar el mismo valor estimado. Durante la segunda fase, intentan tomar una decisión.

Como la mayoría de procesos son correctos, la ejecución del algoritmo no se bloquea, por lo que de esto y de la propiedad del liderazgo eventual del detector de fallas Ω se sigue la propiedad de terminación.

Además, cualquier mensaje $Decide(v)$ contiene un valor $v \neq \perp$, por lo que $\perp$ no puede ser decidido, y v proviene de una variable local est_2 que a su vez proviene de una variable local est_1 que inicialmente contiene valores propuestos por un proceso. Por lo tanto, se cumple la validez.

La propiedad de acuerdo se sigue de que si nos tomamos r la ronda más pequeña en la que un proceso hace broadcast de un mensaje $Decide(v)$, y un algún otro proceso hace broadcast de $Decide(v')$, entonces $v = v'$, y de que el estimado local est_1 de todos los procesos en rondas posteriores será v. Las demostraciones de estas afirmaciones auxiliares se muestran de manera detallada en [3].

$\therefore$ Se puede resolver el consenso utilizando Ω.

Referencias

[1] Chandra, T. & Toueg, S. (1996). *Unreliable Failure Detectors for Reliable Distributed Systems*. Journal of the ACM (JACM), 43(2), 225-267.

[2] Aspnes, J. *Notes on Theory of Distributed Systems*. Yale University, 2017.

[3] Raynal, M. *Fault-Tolerant Message-Passing Distributed Systems. An Algorithmic Approach*. Springer, 2018.