Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida

Tarea 4

Johann Ramón Gordillo Guzmán
418046090

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

3 de Noviembre del 2020

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

Actividades

1. Demuestra que el siguiente protocolo resuelve el consenso en un sistema síncrono donde hasta $f < n$ procesos que pueden fallar.

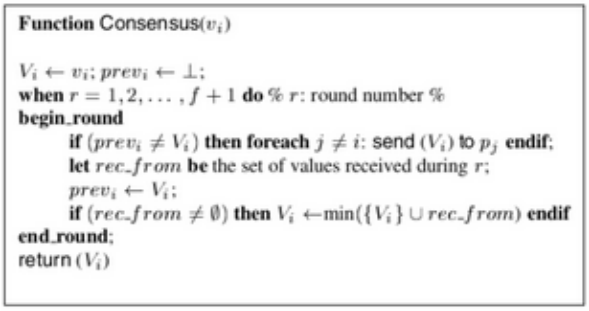


Figura 1: Protocolo de Consenso mejorado

Demostración.

La demostración se divide en tres partes: Probar que hay terminación, validez y acuerdo.

- **Terminación.**

  El algoritmo necesita $f + 1$ rondas para ejecutarse, donde f es el número de procesos que fallan acotado por n. De esta manera, $r = f + 1 \leq n$.

  $\therefore$ El número de rondas está acotado, por lo que una vez que acaban las rondas, el algoritmo termina después del return.

  $\therefore$ Se cumple la terminación.

- **Validez.**

  V_i solo es modificado cuando rec_from es distinto de $\varnothing$, y su valor será el mínimo de su valor anterior y los valores del conjunto rec_from recibido.

  Sin embargo, notemos que rec_from es recibido durante una de las rondas r y su contenido está conformado por los valores enviados por los demás procesos vivos, de tal manera que V_i proviene de algún proceso del sistema.

  $\therefore$ Si un proceso decide V, éste habrá sido propuesto por algún proceso del sistema.

  $\therefore$ Se cumple la validez.

- **Acuerdo.**

  Queremos probar que ningún par de procesos que no fallan decide valores distintos. Es decir, que para todo p_i, p_j procesos del sistema se cumple que después de las $f+1$ rondas tendremos $V_i = V_j$.

Sean entonces $p_i, p_j \in \Pi$ procesos que no fallan, y sea k tal que p_i decide k. Queremos ver ahora que al finalizar la ejecución del algoritmo, p_j también decide k.

Consideramos los siguientes casos, con r el número de ronda tal que a V_i se le da el valor k.

- **Caso 1:** $r < f + 1$.

  Este caso es directo, pues k será el mínimo elemento del conjunto rec_from recibido por p_i, y si c era el valor previo de V_i entonces p_i hará $prev_i = k \neq c$, por lo que p_i enviará $V_i = k$ a p_j en la siguiente ronda, es decir, en $r + 1 \leq f + 1$.

- **Caso 2:** $r = f + 1$.

  En este caso, k fue reenviado en una cadena (en la que ningún proceso aparece más de una vez) de procesos varias veces, desde un proceso p_k hasta p_i, y como a lo más $f < n$ procesos pueden fallar, $\exists p_x \in \Pi$ tal que p_x recibió k en $r' < f + 1$ y por lo tanto lo reenvió en $r' + 1 \leq f + 1$. Es decir p_j recibió k en $r' + 1 \leq f + 1$.

  $\therefore$ No importa el patrón de fallas del sistema, todos tienen que tener lo mismo a final.

$\therefore$ p_i y p_j deciden el mismo valor.

$\therefore$ El protocolo resuelve el consenso en $f + 1$ rondas. ■

2. Argumenta por qué es necesario ejecutar $f + 1$ rondas para llegar a un acuerdo en el protocolo de consenso que ejecuta $f + 1$ rondas.

Function Consensus(v_i)

$V_i \leftarrow [\perp, \ldots, v_i, \ldots, \perp]$; $New_i \leftarrow \{(v_i, i)\}$;
when $r = 1, 2, \ldots, f + 1$ **do** % r: round number %
begin_round
(α) **if** $(New_i \neq \emptyset)$ **then foreach** $j \neq i$: send (New_i) to p_j **endif**;
let $rec\_from[j]$= set received from p_j during r ($\emptyset$ if no msg);
$New_i \leftarrow \emptyset$;
foreach $j \neq i$: **foreach** $(v, k) \in rec\_from[j]$:
(β) **if** $(V_i[k] = \perp)$ **then**
$V_i[k] \leftarrow v$; $New_i \leftarrow New_i \cup \{(v, k)\}$ **endif**
end_round;
let v = first non-$\perp$ value of V_i;
return (v)

Figura 2: Protocolo de Consenso que ejecuta $f + 1$ rondas

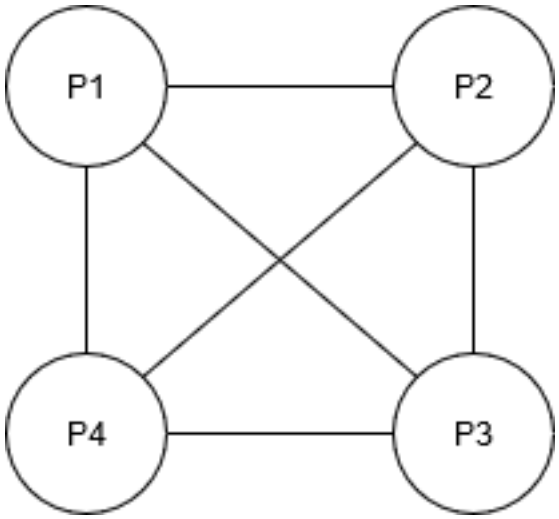
Respuesta.

Consideremos el peor de los casos, en el que tenemos fallas en cascada (o en cadena), donde falla un proceso por ronda.

Al finalizar la ronda $r' = f$, ya habrán sucedido todos los fallos en el sistema, por lo que en la ronda $r = f + 1$ ya no habrá fallos, de manera que todos los procesos vivos tendrán la misma información y podrán tomar una decisión, y por ser un algoritmo donde se cumple el acuerdo, todos tomarán la misma.

Si ejecutaramos únicamente f o menos rondas, no podríamos garantizar que los procesos tuvieran la misma información, por lo que los procesos vivos podrían terminar tomando una decisión distinta y esto sería una contradicción al hecho de que en el algoritmo se cumple el acuerdo. Ejecutar esa ronda extra nos ayuda a garantizar que todos tomen la misma decisión.

Podemos hacer una ejecución para ejemplificar lo anterior:



Supongamos que $f = 2$ y que $P1$ falla en la primera ronda, después de enviar su información a $P2$, y que $P2$ falla en la segunda ronda después de enviar su información a $P3$.

Al finalizar la segunda ronda, ya se garantiza que no habrá fallos en la ronda $f+1 = 3$. Además, en esta ronda que es donde muere $P2$, $P3$ recibe la información particular de $P2$ sumada con la información de $P1$ que $P2$ recibió de $P1$ antes de que muriera.

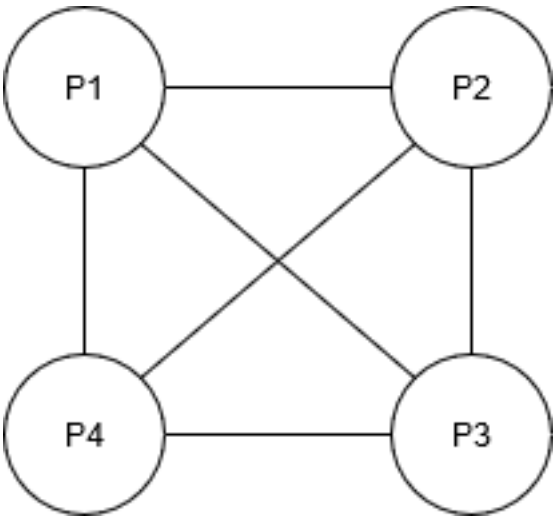
Sin embargo, notemos que $P4$ no ha recibido la información de $P1$ ni de $P2$, por lo que si terminaramos el algoritmo en la ronda $r = f = 2$, $P4$ y $P3$ no tendrían la misma información y terminarían tomando decisiones diferentes, de manera que llegaríamos a una contradicción a la condición de acuerdo del algoritmo.

Así, es necesario que ejecutemos la tercera ronda ($3 = f+1$), para que $P4$ ahora sí tenga la información de $P1$ y $P2$, que será la misma que tiene $P3$ y por lo tanto terminarán tomando la misma decisión, llegando de esta manera a un consenso.

3. Muestra la ejecución del protocolo de la figura 2 en un sistema con $n = 4$ procesos donde $f = 2$ y todos los procesos que fallan lo hacen en la primera ronda sin enviar ningún mensaje antes de fallar.

Respuesta.

Consideremos de nuevo la gráfica $\mathbb{K}_n$ con $n = 4$:



Primero, todos los procesos inicializan sus valores:

Para el proceso $P1$:

$$V_1 = [v_1, \perp, \perp, \perp]$$
$$New_1 = \{(v_1, 1)\}$$

Para el proceso $P2$:

$$V_2 = [\perp, v_2, \perp, \perp]$$
$$New_2 = \{(v_2, 2)\}$$

Para el proceso $P3$:

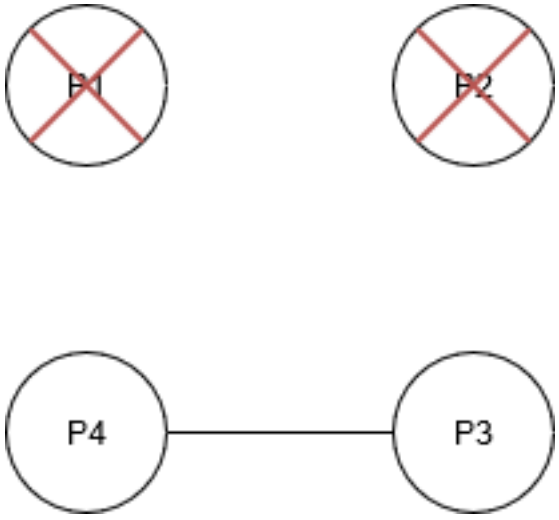
$$V_3 = [\perp, \perp, v_3, \perp]$$
$$New_3 = \{(v_3, 3)\}$$

Para el proceso $P4$:

$$V_4 = [\perp, \perp, \perp, v_4]$$
$$New_4 = \{(v_4, 4)\}$$

En la **Ronda 1**, los procesos $P1$ y $P2$ mueren sin haber enviado ningún mensaje.

La gráfica se ve ahora como sigue:



Sin embargo, $P3$ recibe el mensaje $New_4 = \{(v_4, 4)\}$ de $P4$ y actualiza sus valores:

$$V_3 = [\perp, \perp, v_3, v_4]$$
$$New_3 = \{(v_4, 4)\}$$

Análogamente, $P4$ recibe el mensaje $New_3 = \{(v_3, 3)\}$ de $P3$ y actualiza sus valores:

$$V_4 = [\perp, \perp, v_3, v_4]$$
$$New_4 = \{(v_3, 3)\}$$

En la **Ronda 2**, $P3$ vuelve a recibir un mensaje de $P4$. Esta vez recibe $New_4 = \{(v_3, 3)\}$, pero como $V[3] \neq \perp$, no ejecuta nada y se queda con los siguientes valores:

$$V_3 = [\perp, \perp, v_3, v_4]$$
$$New_3 = \varnothing$$

Análogamente, $P4$ recibe $New_3 = \{(v_4, 4)\}$, pero como $V[4] \neq \perp$, no ejecuta nada y se queda con los siguientes valores:

$$V_4 = [\perp, \perp, v_3, v_4]$$
$$New_4 = \varnothing$$

En la **Ronda 3**, tenemos que $New_3 = New_4 = \varnothing$, por lo que los procesos ya no envían nada y terminamos la ejecución del algoritmo con la elección final de v_3, pues es el primer valor distinto de $\perp$.

4. Si no hay ninguna falla, ¿en cuántas rondas termina el protocolo con early stoping? Justifica tu respuesta.

   Respuesta.

   Si no hay fallas en el sistema, el protocolo con early stopping termina después de **2 rondas**.

   Dentro de esta implementación usamos una bandera *decide*, que cuando es verdadera, indica que ya se puede tomar una decisión y finalizar la ejecución del algoritmo. Si un proceso detecta que ya sabe todo lo que puede saber, levanta la bandera y le avisa a los demás procesos del sistema que está listo para tomar una decisión y finalmente decide, es por ello que necesitamos dos rondas, en una se da cuenta de que ya sabe todo lo que puede saber y en la otra le avisa a los demás procesos.

   Lo anterior funciona ya que la bandera *decide* se levanta cuando el conjunto de procesos que le enviaron mensaje en la ronda anterior es igual al conjunto de procesos que le enviaron mensaje en la ronda actual. Notemos ahora que si suponemos que $R(0) = \Pi$, se cumple que como ningún proceso falla, el conjunto $R(1)$ será igual, por lo que se levantará la bandera *decide* en la primera ronda y el algoritmo finalizará en la segunda ronda despues de que se envíen los mensajes New_i.

Referencias

[1] Raynal, M. *Distributed Algorithms for Message-Passing Systems.* Springer, 2013.

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