Universidad Nacional Autónoma de México
Facultad de Ciencias
Práctica 4 ReadMe
Cervantes Duarte Jose Fernando 422100827
Morales Chaparro Gael Antonio 320076972
Rivera Lara Sandra Valeria 320039823
7 de noviembre de 2024

1 Descripción de Broadcast

Para la implementación del reloj de Lamport usamos como base el algoritmo de Broadcast, y principalmente nos basamos en el siguiente algoritmo:

Algorithm 1 Broadcast con relojes

if $\text{id}_{\text{nodo}} = 0$ **then**
    mensaje $\leftarrow$ datos
    esperar(1 - 5)
    for k en vecinos **do**
        reloj $\leftarrow$ reloj + 1
        eventos.agregar([reloj, 'E'(evento), datos, id_{nodo}, k])
        enviar(datos, reloj, id_{nodo})
    end for
end if
while verdadero **do**
    esperar(1 - 5)
    (datos, reloj_remoto, j) $\leftarrow$ canal_entrada
    mensaje $\leftarrow$ datos
    reloj $\leftarrow$ max(reloj, reloj_remoto) + 1
    eventos.agregar([reloj, 'R'(evento_recibido), datos, j, id_{nodo}])
    esperar(1 - 5)
    for k en vecinos **do**
        if $k \neq j$ **then**
            reloj $\leftarrow$ reloj + 1
            eventos.agregar([reloj, 'E', datos, id_{nodo}, k])
            enviar(datos, reloj, id_{nodo})
        end if
    end for
end while

El método `broadcast` implementa el algoritmo de broadcast utilizando el reloj de Lamport, y consiste en los siguientes pasos:

1.1 Nodo Distinguido (Inicio del Broadcast)

1. Si el nodo es distinguido (con `id_nodo = 0`), inicia el proceso de difusión asignando el mensaje inicial `data` a su variable de instancia `mensaje`.

2. Antes de enviar el mensaje, espera un tiempo aleatorio utilizando la función `env.timeout(randint(1, 5))`, que simula el costo del procesamiento o latencia en un sistema distribuido.

3. El nodo distinguido incrementa su `reloj` de Lamport en +1 antes de enviar el mensaje a cada vecino, asegurando que el evento de envío tenga una marca temporal actualizada.

4. Cada evento de envío se registra en la lista `eventos` con la siguiente estructura:

$$[\text{reloj}, \text{'E'}, \text{data}, \text{id\_nodo}, \text{vecino}]$$

donde:

- `reloj` es la marca temporal actual,
- 'E' indica un evento de envío,
- `data` es el contenido del mensaje,
- `id_nodo` es el nodo emisor, y
- `vecino` es el nodo receptor.

1.2 Recepción y Retransmisión de Mensajes

1. Cada nodo entra en un bucle infinito donde espera la llegada de un mensaje en su `canal_entrada`.
2. Al recibir un mensaje, el nodo extrae el contenido del mensaje, el reloj remoto y el identificador del nodo emisor (`j`).
3. El reloj de Lamport del nodo se actualiza utilizando la fórmula:

$$\text{reloj} = \max(\text{reloj\_local}, \text{reloj\_remoto}) + 1$$

Esto asegura que la marca temporal del evento de recepción es consistente en el sistema distribuido.

4. El evento de recepción se registra en la lista `eventos` con la estructura:

$$[\text{reloj}, \text{'R'}, \text{data}, \text{j}, \text{id\_nodo}]$$

donde:

- `reloj` es la marca temporal del evento,
- 'R' indica un evento de recepción,
- `data` es el contenido del mensaje,
- `j` es el nodo emisor del mensaje, y
- `id_nodo` es el nodo receptor.

5. El nodo espera un tiempo aleatorio antes de retransmitir el mensaje a todos sus vecinos excepto al nodo que le envió el mensaje, excluyendo a `j` de la lista de vecinos.
6. Para cada vecino restante, incrementa su reloj de Lamport y registra el evento de retransmisión en la lista de eventos con la misma estructura que el evento de envío inicial.
7. Finalmente, el mensaje es enviado a los vecinos filtrados utilizando el método `canal_salida.envia`.

2 Actualización del Reloj de Lamport

Cada evento (envío o recepción) incrementa el reloj de Lamport, manteniendo la consistencia temporal en el sistema distribuido. La estructura de eventos en `eventos` facilita la auditoría del orden de eventos en la red y permite analizar la causalidad entre eventos.

3 Descripción DFS con reloj vectorial

3.1 Explicación DFS

Para implementar el DFS con relojes se tuvo que ocupar una lista del tamaño de la cantidad de nodos en la gráfica, por lo cual se le tuvo que decir cuál era la cantidad de nodos desde un inicio a la gráfica.

Para implementar la demora y simular un sistema asíncrono se utilizó la función `randint` y `yield env.time(valor)`, colocándola justo después de recibir un mensaje. Así, cada mensaje recibido sería demorado cierto tiempo antes de poder ser procesado.

En cada envío se agregó el reloj vectorial de ese momento; así, cuando se recibe el mensaje, se toma el mayor reloj y se incrementa en 1 solo en la entrada que corresponde a ese nodo. Cuando se manda tanto "GO" como "BACK", se incrementa en uno el reloj vectorial.

Aparte de mandar el mensaje, el nodo tiene que guardar en su lista de eventos el evento que acaba de ocurrir, ya sea que fuera de envío (E) o de recepción (R). El evento contiene (reloj, tipo de envío, conjunto de visitados, nodo que envía, nodo que recibe). En cada evento, el reloj debe ir incrementando en una de sus entradas. Para su implementación en Python, cuando se manda el reloj a otros nodos, se debe mandar una copia del reloj; de lo contrario, se manda la referencia, y todos los relojes tendrán el mismo valor, provocando que todos los eventos tengan el mismo reloj.

3.2 Cambios respecto a la práctica 3

Por último, respecto al algoritmo DFS que se mandó en la práctica 3, donde cada nodo que era visitado mandaba a sus vecinos que era visitado, ahora en lugar de eso, dentro del mensaje que se envía manda un conjunto de los nodos que se han visitado (incluyéndose a sí mismo) para evitar mandar mensajes extra. Esto es porque ahora no nos conviene que se manden muchos mensajes debido a que el sistema es asíncrono.

3.3 Algoritmo

Algorithm 2 DFS Algorithm with Logical Clocks

Input: N (number of nodes), id_node (unique node ID), V (neighbor set), env (simulation environment)
Initialize $P \leftarrow$ id_node, $H \leftarrow \emptyset$, $R \leftarrow [0, \ldots, 0]$ (logical clock vector)
if id_node $= 0$ **then** ▷ Node is the root
    Set $P \leftarrow$ id_node, $H \leftarrow [V[0]]$
    Increment logical clock $R[$ id_node $] \leftarrow R[$ id_node $] + 1$
    Record event: $(R,' E', \{\text{id\_node}\}, \text{id\_node}, V[0])$
    Send msg: ("GO", {id_node}, id_node, R) to $V[0]$
end if
while True do
    Wait for a random delay
    Receive msg (type, visited, p_j, R_j)
    Synchronize logical clock $R \leftarrow \max(R, R_j)$
    Increment logical clock $R[$ id_node $] \leftarrow R[$ id_node $] + 1$
    Record event: $(R,' R', \text{visited}, p_j, \text{id\_node})$
    if type = "GO" **then**
        $P \leftarrow p_j$
        if $V \subseteq$ visited **then**
            Increment logical clock $R[$ id_node $] \leftarrow R[$ id_node $] + 1$
            Record event: $(R,' E', \text{visited} \cup \{\text{id\_node}\}, \text{id\_node}, P)$
            Send msg: ("BACK", visited $\cup$ {id_node}, id_node, R) to P
        else
            Select unvisited neighbor $s \leftarrow V \setminus$ visited
            $H \leftarrow [s]$
            Increment logical clock $R[$ id_node $] \leftarrow R[$ id_node $] + 1$
            Record event: $(R,' E', \text{visited} \cup \{\text{id\_node}\}, \text{id\_node}, s)$
            Send msg: ("GO", visited $\cup$ {id_node}, id_node, R) to s
        end if
    else if type = "BACK" **then** ▷ Message type is "BACK", return to parent node
        if $V \subseteq$ visited **then**
            if $P =$ id_node **then**
                terminate DFS algorithm
            else
                Increment logical clock $R[$ id_node $] \leftarrow R[$ id_node $] + 1$
                Record event: $(R,' E', \text{visited}, \text{id\_node}, P)$
                Send msg: ("BACK", visited, id_node, R) to P
            end if
        else
            Select unvisited neighbor $t \leftarrow V \setminus$ visited
            $H \leftarrow H \cup \{t\}$ ▷ Add neighbor to list of children
            Increment logical clock $R[$ id_node $] \leftarrow R[$ id_node $] + 1$
            Record event: $(R,' E', \text{visited}, \text{id\_node}, t)$
            Send msg: ("GO", visited, id_node, R) to t
        end if
    end if
end while