Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación distribuida

Práctica 5

Relojes

Baños Mancilla Ilse Andrea - 321173988

Flores Arriola Edson Rafael - 423118018

Rivera Machuca Gabriel Eduardo - 321057608

Reporte de práctica

En esta práctica se implementaron algunos relojes lógicos distribuidos como son el algoritmo de Lamport a través de la clase NodoBroadcast y de Relojes vectoriales con la clase NodoDFS.

NodoBroadcast

Para implementar este algoritmo se usó de base el algoritmo anterior pero se modificaron algunas partes para implementar los relojes lógicos.

Primero agregamos TICK = 1.

Primero revisamos si el nodo es el nodo raíz con id = 0, en este caso hacemos que seen_message sea True, mensaje = data y aumentamos el reloj en 1 pues es una acción de eventos, después por cada vecino en la lista de vecinos del nodo inicial vamos a simular un retraso de 1 a 3 y lo simulamos con timeout, después se tiene que aumentar en 1 el reloj pues es otra acción el envío de mensaje, en este caso por cada mensaje enviado se agrega 1 al reloj por cada vecino.

Si no era el nodo raíz entonces entra a while, ahí se tiene que desempaquetar el mensaje y vamos a revisar que el tipo de mensaje sea BROADCAST y que seen_message sea False, en ese caso el nodo que recibió el mensaje va a actualizar su reloj, este se actualiza tomando su reloj lógico y el del nodo que le envió el mensaje, de estos dos va a tomar el máximo y después a a aumentar en 1, después seen_message se hace true, mensaje = info_mensaje que es la data del nodo raíz, agregamos el evento y finalmente hacemos lo mismo que el nodo raíz de enviar mensaje a todos sus vecinos simulando un retraso, sin embargo, en este caso habŕa una modificación la cual es que solo se envía el mensaje si y sólo sí el vecino es diferente al emisor del mensaje, es decir, que no le manda el mensaje a quien anteriormente se lo mando.

NodoDFS

Para este algoritmo se usó de base el algoritmo anterior pero aún así se hicieron algunas modificaciones.

Primero agregamos TICK = 1, también hicimos que reloj sea un vector del tamaño de todos los nodos para hacer los relojes lógicos.

Después definimos unas funciones que se usarán en cada nodo después de alguna acción. Son las siguientes:
aumenta_reloj(self). Esta función lo único que hace es que al nodo que la llame su reloj aumentará en 1.
actualiza_reloj(self, reloj). Esta función toma el reloj del nodo que lo llama y recibe un reloj de un nodo que le mando un mensaje, su función es actualizar el reloj del nodo que la llama para que los tiempos de todos los nodos sean siempre el máximo.
registra_evento(self, tipo, mensaje, emisor, receptor). Esta función se encarga de agregar a los eventos que suceden guardando una copia del reloj, el tipo de evento, el mensaje, el emisor y el receptor.

Al inicio del algoritmo primero revisamos al nodo 0 e hicimos que el mismo sea su propio padre e inicializamos sus hijos y completed_children. Si es el nodo 0 entonces entramos a sus vecinos y tomamos el primer vecino, a este se le manda el el mensaje tipo GO, None y su propio id para que el receptor sepa quien es el emisor. Además se agrega k a la lista de completed_children y se aumenta el reloj del nodo 0.

Ahora entramos a la simulación de envios de mensajes y la propagación de los mismos.
El que recibe el mensaje desempaqueta el mensaje y actualiza su reloj usando el reloj del emisor y después aumenta su propio reloj, también registra el evento.

Guardamos el emisor del mensaje como j para usarlo después.

Revisamos el tipo de mensaje, tenemos dos tipos, GO y BACK

- GO

  En este caso revisamos si el nodo ya tiene padre:

  - No tiene

    En este caso el emisor j es el padre del que recibio el mensaje además agregamos a completed_children a j y con esto filtramos a los vecinos del nodo receptor para quitar a j y no mandarle de nuevo mensaje GO.

    Si el nodo ya no tiene vecinos, entonces vamos a aumentar su reloj y empieza la propagación hacia arriba, también registramos ese evento y le mandamos el mensaje a su padre.

Si el nodo todavía tiene vecinos, entonces tomamos el primer vecino de la lista y lo agregamos a completed_children, también aumentamos el reloj del nodo y seguimos la propagación del mensaje de GO hacia abajo mandándolo a su hijo.

- Tiene

  En caso de que tenga padre entonces aumentamos su reloj y vamos a regresar el mensaje de no.

- BACK

  En este caso vamos a revisar el tipo de respuesta, si la respuesta es yes entonces guardamos al nodo emisor como hijo, pues esto ya va de subida, además agregamos a completed_children. También vamos a filtrar a los vecinos restantes para quitar a los vecinos que ya estén en completed_children. Ahora vamos a revisar si el nodo todavía tiene vecinos no visitados:

  - No tiene

    Revisamos si su propio padre, entonces es el nodo raíz y terminamos.

    De no ser su propio padre entonces aumentamos su reloj y vamos a seguir propagando el mensaje hacia arriba con tipo BACK y respuesta yes. Este mensaje se le va a mandar al padre.

  - Tiene

    En caso de que tenga es porque ya había llegado a una hoja y subió, entonces vamos al otro lado de la hoja siguiendo los mismos pasos, tomamos a k como el primer vecino en la lista de los vecinos filtrados y lo agregamos a completed_children, aumentamos su propio reloj y mandamos el mensaje GO, además registramos el evento.