Computación Distribuida - Práctica 2

Equipo:

- Alejandro Axel Rodríguez Sánchez - ahexo@ciencias.unam.mx
- José David Aguilar Uribe - jdu@ciencias.unam.mx

Version de Python: 3.13.2

Introduccion

En este reporte detallamos la implementación de una simulación de un sistema distribuido síncrono para construir un árbol BFS (Breadth-First Search) utilizando Python y la biblioteca SimPy. El objetivo es modelar cómo los procesos en una red distribuida colaboran para construir un árbol BFS de manera síncrona, partiendo de un proceso distinguido seleccionado por el usuario.

Desarrollo

En esta práctica, hemos implementado un sistema distribuido síncrono que construye un árbol BFS (Breadth-First Search) a partir de un proceso distinguido. Para lograrlo, utilizamos la biblioteca SimPy de Python, que nos permite modelar el paso del tiempo y la sincronización entre procesos en un entorno simulado. A diferencia de la Práctica 1, donde se implementó el algoritmo MergeSort, en esta ocasión nos enfocamos en la construcción de un árbol BFS, un problema clásico en computación distribuida que requiere comunicación y coordinación entre procesos.

Comportamiento de los procesos

Cada proceso en el sistema sigue un conjunto de reglas bien definidas para participar en la construcción del árbol BFS:

1. **Inicialización:** Cada proceso comienza sin un padre asignado y sin un nivel definido en el árbol.
2. **Recepción de mensajes:** Los procesos intercambian dos tipos de mensajes:
    - "GO": Indica que un proceso vecino intenta reclutarlo como hijo en el árbol BFS.
    - "BACK": Es una respuesta que indica si el proceso aceptó o rechazó la invitación.
3. **Procesamiento de mensajes "GO":**
    - Si el proceso no tiene padre, acepta al remitente como su padre y calcula su nivel en el árbol (distancia al root + 1).
    - Si ya tiene un padre pero el nuevo mensaje ofrece un nivel más bajo (es decir, un camino más corto al root), actualiza su padre al nuevo remitente.
    - Si no puede aceptar al remitente como padre (porque ya tiene uno con un nivel igual o menor), responde con un mensaje "BACK" de rechazo.

4. **Envío de mensajes "GO":** Tras aceptar un padre, el proceso envía mensajes "GO" a todos sus vecinos (excepto al padre) para intentar reclutarlos como hijos.
5. **Procesamiento de mensajes "BACK":**
    - El proceso espera respuestas "BACK" de los vecinos a los que envió mensajes "GO".
    - Si un vecino acepta ser hijo (respuesta "yes"), lo agrega a su lista de hijos.
    - Una vez recibidas todas las respuestas, el proceso verifica si ha terminado su parte en la construcción del árbol.
    - Si es el proceso distinguido y ha recibido todas las respuestas, anuncia que el árbol BFS está completo.

Este comportamiento asegura que el árbol se construya nivel por nivel, comenzando desde el proceso distinguido, y que cada proceso esté conectado al padre que le ofrece el camino más corto al root, cumpliendo las propiedades de un árbol BFS.

Implementacion

La implementación utiliza SimPy para simular un sistema distribuido síncrono, donde los procesos operan en rondas discretas. A continuación, detallamos cómo se estructuró el código y se manejó la sincronización.

Estructura del codigo

1. **Stores:**
    - Cada proceso tiene un `simpy.Store` que funciona como su buzón de mensajes, permitiendo el envío y recepción asíncrona de mensajes.
2. **Funcion `envia_msg`:**
    - Simula el envío de un mensaje ("GO" o "BACK") entre procesos.
    - Incluye un retraso `(env.timeout(peso))` basado en el peso de la arista entre el remitente y el receptor, extraído de la matriz de adyacencias.
3. **Funcion `proceso`:**
    - Define el comportamiento de cada proceso.
    - Procesa mensajes "GO" y "BACK" según las reglas descritas, actualizando el árbol y enviando mensajes cuando corresponde.
    - Usa `yield env.timeout(1)` para simular el paso de una ronda antes de procesar el siguiente mensaje.
4. **Grafica del sistema:**
    - Representa la topología de la red como una matriz 6x6, donde cada entrada indica el peso de la arista entre dos procesos (0 si no hay conexión).

Manejo de la sincronización

- Los procesos operan en rondas discretas, esperando 1 unidad de tiempo simulado entre acciones mediante `yield env.timeout(1)`.

- Los mensajes se envían y reciben de forma síncrona, asegurando que todos los procesos avanzan al mismo ritmo.
- El retraso en la transmisión de mensajes (basado en los pesos de las aristas) se modela con `env.timeout(peso)`.

Ejecucion

Ejecutar el script:

```
python3
Practica2_AlejandroAxelRodriguezSanchez_JoseDavidAguilarUribe.py
```

El programa solicita al usuario elegir un proceso distinguido (un número entre 0 y 5):

Ejemplo de ejecucion

```
$ python3
Practica2_AlejandroAxelRodriguezSanchez_JoseDavidAguilarUribe.py
Escoge el proceso distinguido(0-5): 0
Ronda 1: p0 se convierte en padre de p0
Ronda 2: p0 se convierte en padre de p2
Ronda 3: p0 se convierte en padre de p1
Ronda 3: p2 se convierte en padre de p4
Ronda 4: p4 se convierte en padre de p3
Ronda 5: p2 se convierte en padre de p3
Ronda 5: p3 se convierte en padre de p5
Ronda 6: p3 se convierte en padre de p5
p0 El arbol bfs se ha construido
parents = {0: 0, 1: 0, 2: 0, 3: 2, 4: 2, 5: 3}
levels = {0: 0, 1: 1, 2: 1, 3: 2, 4: 2, 5: 3}
```

2025-2, Grupo 7106.

Profesor: Mauricio Riva Palacio Orozco.

Ayudante: Adrián Felipe Fernández Romero.

Ayudante de laboratorio: Daniel Michel Tavera.