Proyecto final - Búsqueda del tesoro

Computación paralela y cálculo distribuido

Mario García Mayo

1. Descripción del juego

En una isla, representada por una cuadrícula, hay un tesoro escondido en una casilla y cuatro jugadores, jugando por parejas, tienen que tratar de encontrarlo. Cada turno, un jugador se mueve a una casilla adyacente. Los miembros de cada pareja se van comunicando para no pisar las mismas casillas. El juego termina cuando algún jugador encuentra el tesoro y su pareja se declara ganadora.
A la hora de implementar el juego, cada uno de los jugadores será un nodo distinto y además habrá un nodo máster que hará los preparativos de la partida y conocerá la localización del tesoro.
La topología de la red es parecida a una topología estrella, pero los extremos también se comunican entre sí dos a dos:

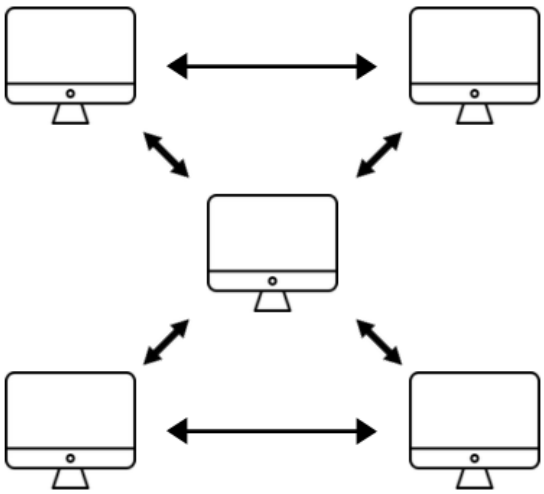

Figura 1: Topología de la red

2. Desarrollo del juego

Al comienzo de la partida, el nodo máster divide el tablero en cuatro cuadrantes y coloca a un jugador en cada uno de ellos. Después, coloca el tesoro en una casilla aleatoria donde no haya ningún jugador. Todo esto se hace con la función `generar_juego`:

```
def generar_juego(m,n):
    #Primer cuadrante - J1
    j1_x=random.randint(0,n//2-1)
    j1_y=random.randint(0,m//2-1)
    j1=j1_x+j1_y*n

    ... #El resto de cuadrantes son igual

    #Tesoro
    tesoro=random.randint(0,n*m-1)
    while tesoro==j1 or tesoro==j2 or tesoro==j3 or tesoro==j4:
        #Nos aseguramos de que el tesoro no este en la misma casilla
        que un jugador
        tesoro=random.randint(0,n*m-1)

    return [tesoro,j1,j2,j3,j4]
```

Una vez que el nodo ha generado el juego, debe comunicar a cada jugador las dimensiones del tablero y sus posiciones correspondientes. Las dimensiones son datos que deben conocer todos los jugadores así que el envío se hace con `bcast`, mientras que la posición de cada jugador a priori solo la conoce él mismo luego el envío se hace con `scatter`.
A continuación, los miembros de cada pareja comparten sus posiciones iniciales. Los envíos serán punto a punto y se realizarán con `send` y `recv`. Cada jugador guarda las posiciones visitadas por sí mismo y por su compañero en el array `pisadas` utilizando `.append()`.

Una vez alcanzado este punto, comienzan los turnos de la partida. Cada nodo tiene una variable `fin_juego` cuyo valor indica si la partida debe terminar, y para ejecutar los turnos hasta que esto suceda el resto de código estará dentro de un bucle

`while fin_juego == 0`.
Para moverse, los jugadores utilizarán la función `movimiento`:

```
def movimiento(m,n,x,pisadas):

    direcciones=[-1,1,-n,n]# izquierda,derecha,arriba y abajo

    #ordenamos aleatoriamente las direcciones
    random.shuffle(direcciones)
    #Direccion principal: sera una QUE NO HAYAMOS PISADO (si existe,
    si no se quedara en 0)
    dir1=0
    #Direccion secundaria: sera una A LA QUE ES FISICAMENTE POSIBLE
    MOVERSE (NO BORDES), por si hemos pisado todas las de alrededor
    dir2=0
    i=0

    #Bucle para movernos: comprobamos si hay alguna casilla adyacente
    que no hayamos pisado ya. Si la encontramos, la guardamos en dir1
    y salimos del bucle. Si no, guardamos una direccion aleatoria en
    dir2 y salimos del bucle (i==4)
    while i!=4 and (dir1 not in direcciones):
        direccion=direcciones[i]
        #SALE DIRECCION IZQUIERDA
        # Sale direccion izquierda y no estamos en el borde
        if direcciones[i]==-1 and x%n!=0:
            dir2=direcciones[i]
            # No hemos pisado la casilla de la izquierda aun
            if x-1 not in pisadas:
                dir1=direcciones[i]

        elif ... #Igual para el resto de direcciones

        i=i+1

    #Si hemos salido del bucle y dir1 es 0, significa que no hemos
```

```
encontrado una direccion que no hemos pisado asi que vamos en una
aleatoria
if dir1==0:
    x=x+dir2
#Si no, significa que hemos encontrado una direccion que no hemos
pisado luego nos movemos a esa y añadimos la casilla a la lista de
pisadas
else:
    x=x+dir1
    #Apuntamos la nueva casilla no pisada
    pisadas.append(x)

return x
```

Cuando un jugador de la pareja se mueve, envía a su pareja su nueva posición con `send` y `recv` y esta la registra en la lista de pisadas con `.append()` (el registro de las casillas que pisa un mismo jugador va implícito en la función `movimiento`). Una vez actualizada su lista, el segundo miembro de la pareja se mueve y envía su posición al primero del mismo modo para que la registre.
Cuando las dos parejas llegan a este punto, el nodo máster recoge todas las posiciones con un `gather` para ver si algún jugador ha encontrado el tesoro. En caso negativo, no ocurre nada y la partida continúa. En caso afirmativo, el nodo máster registra en `fin_juego` quién ha ganado y mediante `bcast` hace saber a todos los jugadores quién es el ganador a la vez que les da la instrucción de dejar de jugar.
A la hora de hacer las comprobaciones, el nodo máster va mostrando por pantalla todas las posiciones de los jugadores para poder realizar un seguimiento del juego. También muestra por pantalla quién es el ganador cuando el juego termina.

**En el archivo .py se va explicando cada línea del código con comentarios.*

3. Operaciones colectivas y punto a punto

A lo largo del programa, se realizan envíos tanto punto a punto como colectivos. Los envíos punto a punto los realizan los miembros de cada pareja para comunicarse entre sí, mientras que los colectivos se realizan para la comunicación del máster con los jugadores.

Respecto a la posibilidad de intercambiar operaciones colectivas y punto a punto, en el caso de las colectivas se podría realizar este intercambio y que el nodo central se fuese comunicando de uno en uno con cada jugador, aunque las operaciones colectivas son una forma mucho más compacta de hacerlo.
Respecto a las comunicaciones punto a punto de cada pareja, si se hiciesen colectivas con un `bcast` se estaría dando información a la pareja enemiga, luego no tendría mucho sentido. Se podría hacer con un `scatter` enviando una variable vacía (o con un valor que no tenga que ver con el juego) a los miembros de la pareja enemiga, pero resultan mucho más naturales las operaciones punto a punto en este caso.

4. Multiprocessing o threading en cada nodo

Respecto a la opción de implementar paralelismo en algún nodo para agilizar la ejecución del programa, en nuestro caso no tiene mucho sentido hacerlo. Cuando las dimensiones del tablero se volviesen muy grandes, que es cuando se podría aprovechar la ejecución multihilo, la operación donde tendría sentido implementar multihilo sería en la búsqueda de posiciones ya visitadas en la lista `pisadas`.
Sin embargo, las dimensiones del tablero tendrían que ser muy grandes para que realmente se notase alguna mejora. La lista `pisadas` comienza teniendo un tamaño muy pequeño y en dimensiones bajas no llega a alcanzar un tamaño excesivamente grande, luego implementar búsqueda multihilo en una lista pequeña solo ralentizaría el juego al tener que crear y asignar trabajo a distintos procesos.