Computación Distribuida - Práctica 1 [*]

Alejandro Axel Rodríguez Sánchez [†] José David Aguilar Uribe [‡]

5 de marzo del 2025

Resumen

En este reporte detallamos la implementacion de una sencilla simulación de un sistema distribuido síncrono con el propósito de resolver el algoritmo de ordenamiento *MergeSort*, usando Python3K y la biblioteca SimPy.

Introducción

Desarrollo

Como SimPy únicamente provee un conjunto de directrices y objetos para abstraer los eventos discretos y el paso del tiempo, el resto del modelaje de la simulación es muy atribrario y se deja completamente a las preferencias o necesidades del desarrollador.

En nuestro caso, hay dos soluciones prágmáticas que se elucidan inmediatamente:

1. **Modelar todo el sistema en un paradigma orientado a objetos**, escribir una clase para modelar a los nodos de la red distribuida, poner métodos para cada operación interna como mezclar u ordenar, para finalmente construir el sistema con instancias de esta y correr la simulación.
2. **Modelar únicamente las partes funcionales del sistema**, es decir, definir únicamente el comportamiento de los nodos con un método el cual se ejecuta y emite salidas que daría un proceso en la red distribuida. No modelamos todos los procesos, sólo lo que hacen.

Para esta práctica, hemos optado por la segunda opción.

Comportamiento de los procesos

Cada proceso en el sistema a modelar sigue la siguiente rutina:

[*] 2025-2, Grupo 7106. Profesor: Mauricio Riva Palacio Orozco. Ayudante: Adrián Felipe Fernández Romero. Ayudante de laboratorio: Daniel Michel Tavera.

[†] ahexo@ciencias.unam.mx

[‡] jdu@ciencias.unam.mx

1. Recibe una lista ordenable.
2. Verifica si no es que cumple el caso trivial (que tenga a lo mas 2 elementos).
3. Si es del caso trivial, la ordena trivialmente: Si tiene dos elementos, la voltea por el más grande y regresa la lista. Si tiene uno o ninguno, la regresa tal cual.
4. Si la lista no es trivial, la parte en dos e invoca a otros dos sub-procesos a los que les pasa cada mitad.
5. Espera a obtener las listas ordenadas de sus sub-procesos (que seguirán la misma rutina).
6. Cuando reciban las sub listas ordenadas, las unen en una sola. Esta operación también es trivial, pues sabemos de antemano que las listas están ordenadas, así que basta iterar sobre de ellas a la vez e ir colocando el elemento más grande primero en la lista definitiva.
7. Regresan esta lista y concluyen.

Esta rutina la hemos implementado concretamente en Python del siguiente modo, donde además consideramos que todos los procesos tienen un identificador entero único:

```python
def ordenar(lista: list, id: int):
# PASO 1 al 3
    if len(lista) == 2:
        return lista if lista[0] < lista[1] else lista[::-1]
    elif len(lista) == 1:
        return lista
# PASO 4
    mitad = len(lista) // 2
    id_hijo_izq = id * 2
    id_hijo_der = id * 2 + 1
# PASO 5
    lista_izq = ordenar(lista[mitad:], id_hijo_izq)
    lista_der = ordenar(lista[:mitad], id_hijo_der)
# PASO 6
    lista_ordenada = []
    i, j = 0, 0

    while i < len(lista_izq) and j < len(lista_der):
        if lista_izq[i] <= lista_der[j]:
            lista_ordenada.append(lista_izq[i])
            i += 1
        else:
            lista_ordenada.append(lista_der[j])
            j += 1
    lista_ordenada.extend(lista_izq[i:])
    lista_ordenada.extend(lista_der[j:])
# PASO 7
    print(f"[Ronda {ambiente.now}] BACK: Desde {id}")
    return lista_ordenada
```