Desarrollo de un sistema distribuido con Zookeeper

Carlos García Guzmán
Universidad de Málaga
carlosgarciag552@uma.es

Contents

1. Funcionamiento básico del sistema 1
   1.1. Lógica de la Aplicación 1
   1.2. Funcionalidad del Líder 1
   1.3. Robustez 1
   1.4. Ejemplo de ejecución inicial 2

1. Funcionamiento básico del sistema

Se ha desarrollado una solución en Python `src/app.py` que orquesta un conjunto de nodos sensores coordinados mediante **Apache ZooKeeper**, concretamente a través de la librería `kazoo` disponible para Python.

El sistema se compone de un conjunto de nodos que se comunican con un servidor **ZooKeeper** para coordinar sus acciones. Cada nodo se identifica con un ID único y realiza dos funciones principales: actuar como sensor y participar en un proceso de elección de líder.

1.1. Lógica de la Aplicación

Tal como se acaba de mencionar, cada nodo/instancia de la aplicación realiza dos funciones concurrentes:

1. **Sensor (Thread secundario)**: Genera mediciones aleatorias cada 5 segundos y las publica en ZooKeeper. Se utilizan **znodes efímeros** en la ruta `/mediciones/app{id}`, asegurando que las mediciones de un nodo solo existan mientras dure la sesión de ese nodo.
2. **Coordinador/Elección (Thread principal):** Utilizando la receta `Election` de la librería `kazoo`. El nodo escogido como líder es el encargado de recolectar los datos que escriben todos los nodos (incluido él) y enviarlos a la API desarrollada en la práctica anterior.

1.2. Funcionalidad del Líder

El líder ejecuta un bucle infinito donde:

- Recupera la lista de nodos hijos en `/mediciones`.
- Lee el valor de cada nodo, gestionando posibles condiciones de carrera (nodos que se desconectan durante la lectura).
- Calcula la media aritmética de los valores disponibles.
- Envía el resultado a una API externa `http://localhost/nuevo?dato=valor`
- Envía el resultado a una API externa `http://localhost:8080/nuevo?dato=valor`

1.3. Robustez

El código incluye manejo de señales (`SIGINT`) para un cierre ordenado de la conexión con ZooKeeper y bloques `try-except` para gestionar errores de red al comunicar con la API o inconsistencias temporales en los datos de ZooKeeper.

1.4. Ejemplo de ejecución inicial

Antes de probar la correcta ejecución del sistema nos debemos asegurar de que tenemos un contenedor ejecutando Zookeeper, así como otro que ejecute la API desarrollada en la práctica anterior (por simplicidad, en esta sección usaremos el fichero **`compose-inicial.yaml`**, el cual se encarga de ejecutar tanto nuestra API como zookeeper):

```
docker compose -f compose-inicial.yaml up
```

Si ahora ejecutamos varias instancias de nuestra app podemos observar que una de ellas es escogida como lider, como se puede observar en Figure 1:

Figure 1: Ejecución simple de $N = 2$ instancias de nuestra aplicación.

Si ahora interrumpimos al lider, podemos ver que automáticamente la otra instancia de la aplicación es elegida como nuevo líder, como se muestra en Figure 2

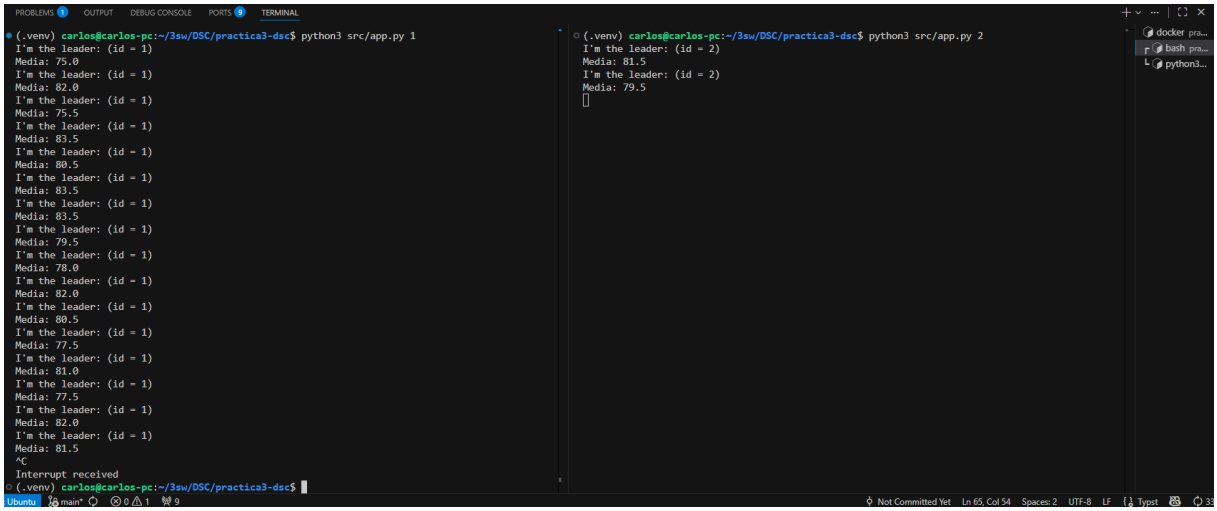

Figure 2: Interrupción del lider