POYECTO ANÁLISIS DE DATOS UMBRELLA CORPORATIÓN

Se creo un programa con la finalidad de que sea capaz de simular un alto volumen de datos, que pueden provenir de distintos sitios simultáneamente y este los pueda procesar y convertir a un mismo tipo de dato. El programa fue desarrollado utilizando Python con una arquitectura de concurrencia.

Dependencias del programa:

Para ejecutar el programa se necesita tener instalado las de pendencias de pandas, matplotlib y diagrams.

pip install pandas matplotlib diagrams

Flujo de Procesamiento de Datos

BioServices, FisicService, GeneticService, estas clases son las que envían los datos. Estas tienen una función async que espera a su turno para insertar raw_data.

Luego manda los datos a un normalizador, donde se validan y se transforman a un formato dict estándar para que el resto del sistema pueda entenderlo.

El main principal funciona de la siguiente manera.

Primero se crea la cola principal de trabajo (queue), se obtienen el número de núcleos del procesador para cantidad de procesos paralelos que puede se pueden usar y se se crean los recursos compartidos entre procesos (share_results y lock).

Cada servicio (BioServices, FisicService, GeneticService) ejecutan un producer() asíncrono que llama a fech_data y coloca cada dato en la cola principal. Se evalúa también si el dato es crítico (manda alerta si lo es).

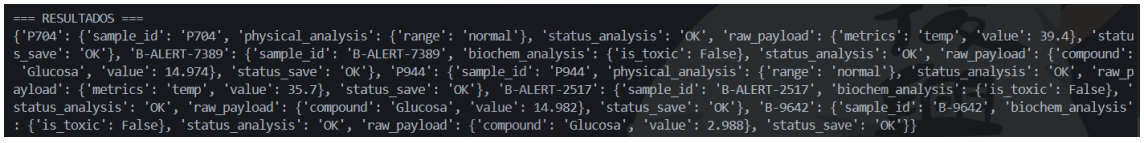
Luego los consumidores (rango ajustable está en 3), están pendientes de la cola principal para procesar datos según su tipo. Si es genético se manda al pool de procesos para ser analizado usando paralelismo de la CPU (analyze_genetic_data en el workers.py).

Si es bioquímico o físico, se analizan con funciones asíncronas que simulan tareas de I/O (analyze_biochem_data, analycze_physical_data en el worker.py).

Los resultados se guardan en shared_results, usando el lock para evitar conflictos.

Luego entramos en la fase de apagado donde el programa cancela todos los productores, espera a que la cola se vacíe y imprime un resumen del resultado con los datos.

Ej:



Por ultimo se genera un informe final que crea un DataFrame con métricas.

Calcula: Latencia, Numero de operaciones exitosas / fallidas y la latencia de las alertas.

Se muestran las alertas criticas que han ido surgiendo y se genera un grafico en png que representa la evolución de las latencias a lo largo del tiempo.

Ej:

```
=== INFORME FINAL DE MÉTRICAS ===

1. Tiempo de respuesta medio (ms) por tipo:
type
BIOQUIMIC     221.903287
FISIC         220.952919
GENETIC      1063.385063

2. Log de Alertas Críticas Procesadas:
Se procesaron un total de 34 alertas críticas:
       sample_id       type     timestamp                                payload_data
0           P704      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 39.4}
1   B-ALERT-7389  BIOQUIMIC  1.762799e+09  {'compound': 'Glucosa', 'value': 14.974}
2   B-ALERT-2517  BIOQUIMIC  1.762799e+09  {'compound': 'Glucosa', 'value': 14.982}
3   B-ALERT-7934  BIOQUIMIC  1.762799e+09  {'compound': 'Glucosa', 'value': 14.857}
4           P138      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 40.6}
5           P345      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 39.8}
6           P850      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 40.6}
7           P719      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 39.3}
8           P777      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 40.9}
9           P872      FISIC  1.762799e+09        {'metrics': 'temp', 'value': 39.2}
```

```
3. Latencia a la detección y envío de alertas (ms):
Latencia media de alertas: 3.50 ms

4. Tasa de errores / fallos:
Operaciones totales: 144
Fallos totales: 0
Tasa de fallos por millón de operaciones: 0.00

5. [Visual] Tabla de eventos procesados por tipo:
           total_procesados  latencia_media_ms
type
BIOQUIMIC                51         221.903287
FISIC                    66         220.952919
GENETIC                  27        1063.385063
```