Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2523 - Sistemas Distribuídos

Tarea 3

15 de diciembre de 2020

2º semestre 2020 - Profesor Cristián Ruz

Maximiliano Schudeck - 16638530

Queues Asíncronas con Redis y RQ.

Ejecución

Para poder ejecutar correctamente el programa, se debe tener una instancia de Redis corriendo en localhost puerto 6379. Se ha incluído un archivo docker-compose, el cual permite, si se tiene instalado docker y docker-compose, con los siguientes comandos, lograr tener una instancia funcional de Redis. Se creará en el puerto 6379 por default.

```
docker-compose build
docker-compose up
```

Luego, debemos crear el worker de rq. Una vez tenemos nuestro Redis funcionando en el puerto default, debemos correr en una terminal nueva, el comando:

```
rq worker high medium low
```

Lo cual creará un worker para nuestro Redis, con 3 queues de prioridad, de manera que cuando le toque ejecutar nuevamente un comando, revisará primero la queue "high", luego "medium", y luego "low".

Finalmente, en una nueva terminal, es necesario correr nuestro archivo python principal, el cual se encargará de leer el archivo 'instruction_file.txt' ubicado en la misma carpeta de 't3.py', conectarse a la instancia de Redis, crear un archivo output.txt vacío, usará las queues establecidas en el worker, y se encargará de leer el archivo y de encolar en el worker, el cual escribirá una linea en el archivo output.txt cuando termine la ejecución de cada función.

```
python t3.py instruction_file.txt
```

Funcionamiento asíncrono, encolamiento y output.

El programa principal se encarga, principalmente, de conectarse con Redis, y de encolar cada ejecución del programa dependiendo de si el cliente es VIP, Alto o Común, en las queues `"high"`, `"medium"`, and `"low"`, respectivamente.

En el archivo `functions_hash.py`, podemos encontrar las 2 funciones entregadas por el enunciado, así como una 3ra función la cual utilizo para escribir la linea correspondiente en el archivo `output.txt`, la cual se llama `proof_of_work_with_client()`.

Podemos comprobar el funcionamiento asíncrono con un output de prueba.

```
Linea: 0    Cliente: C1    k':4028373
Linea: 1    Cliente: T1    k':4836549
Linea: 3    Cliente: T0    k':2548241
Linea: 2    Cliente: C0    k':7322412
```

Acá usé el input de prueba entregado en el github de la tarea, el cual se encuentra en mi entrega con el nombre de `instruction_file2.txt`. Podemos ver como lo primero que hace es ejecutar la primera instrucción, ya que es la primera que entra a cualquiera de las queues, y luego mientras continúa ejecutando la primera instrucción, llegan a la cola las instrucciones de mayor prioridad(cola "medium", cliente tipo T), las cuales proceden a ser encoladas y son ejecutadas primero que las de baja prioridad(cola "low", correspondiente a los clientes tipo C).

Quise mostrar las lineas leídas, para que se viera que lo que importa dentro de una queue NO es el número ID del cliente, si no el orden en que es leído y éste entra a la queue.

Dado que cada cliente ejecuta la misma acción, lo único que cambia entre un cliente VIP, Alto y Común, es la cola en la cuales sus instrucciones serán encoladas. Así, el cliente C0 es ejecutado después de los clienes T1 y T0, a pesar de que T0 fue encolado después de C0, esto ya que T0 se encontraba encolado en la cola `'medium'`, y C0 en la cola `'low'`.

El output obtenido siempre se llamará `output.txt`. Si no existe, será creado, si ya existe un archivo con ese nombre, se sobreescribirá