Multithreading avec ThreadPool en Python

Net and Light

2023 - 2024

Contents

1 Introduction

Le multithreading permet d'exécuter plusieurs opérations simultanément, ce qui peut considérablement améliorer les performances des applications qui effectuent des tâches IO-bound, comme les requêtes réseau. En Python, le module `multiprocessing.dummy` fournit une implémentation du `ThreadPool`, qui permet d'exécuter des fonctions en parallèle à l'aide de threads.

1.1 Code

```python
from multiprocessing.dummy import Pool as ThreadPool

def fetch_data(ip):
    # Fonction qui récupère des données à partir d'une adresse IP
    pass

ip_addresses = {'ip1': '192.168.1.1', 'ip2': '192.168.1.2'} # Exemple d'adresses IP

pool = ThreadPool(len(ip_addresses)) # Crée un pool de threads avec une taille égale au nombre d'adresses IP
pool.map(fetch_data, ip_addresses.values()) # Applique la fonction fetch_data à chaque adresse IP en parallèle
pool.close() # Ferme le pool
pool.join() # Attends que toutes les tâches soient terminées
```

1.2 Explication du code

- `from multiprocessing.dummy import Pool as ThreadPool`: Nous importons `ThreadPool` depuis le module `multiprocessing.dummy`.
- `def fetch_data(ip)`: La fonction qui sera exécutée par chaque thread. Elle prend une adresse IP en argument et effectue une tâche, comme récupérer des données depuis cette adresse.
- `ip_addresses`: Un dictionnaire contenant des adresses IP comme exemple.
- `pool = ThreadPool(len(ip_addresses))`: Nous créons un `ThreadPool` avec un nombre de threads égal au nombre d'adresses IP. Chaque thread pourra exécuter une tâche en parallèle.
- `pool.map(fetch_data, ip_addresses.values())`: La méthode `map` applique la fonction `fetch_data` à chaque adresse IP simultanément. Les tâches sont réparties entre les threads du pool.
- `pool.close()`: Nous fermons le `ThreadPool` pour empêcher l'ajout de nouvelles tâches.
- `pool.join()`: Nous attendons que toutes les tâches en cours soient terminées avant de continuer.

2 Fonctionnement du multithreading

Le multithreading permet de diviser une tâche en plusieurs sous-tâches qui peuvent être exécutées simultanément par des threads distincts. Cela est particulièrement utile pour les opérations IO-bound, comme les requêtes réseau, où les threads peuvent attendre des réponses sans bloquer l'exécution des autres tâches.

2.1 Avantages

- **Amélioration des performances**: Le multithreading peut réduire le temps total d'exécution en exécutant des tâches en parallèle.
- **Meilleure utilisation des ressources**: Les threads peuvent utiliser des périodes d'attente pour effectuer d'autres tâches, améliorant ainsi l'efficacité globale.

2.2 Inconvénients

- **Complexité accrue**: La gestion des threads peut compliquer le code et introduire des erreurs difficiles à détecter, comme les conditions de course.
- **GIL (Global Interpreter Lock)**: En Python, le GIL peut limiter les gains de performance pour les tâches CPU-bound en ne permettant qu'à un seul thread d'exécuter du code Python à la fois.

3 Conclusion

Le multithreading avec `ThreadPool` est un moyen efficace de paralléliser les tâches IO-bound en Python. En utilisant un pool de threads, nous pouvons exécuter plusieurs tâches simultanément, réduisant ainsi le temps total d'exécution et améliorant les performances de l'application.