Traitement des données distribuées

Multiprocessing vs multithreading

ENSAE 2020/2021

Mastère Spécialisé - Data Science

Lilia BEN BACCAR - Amale NOKRI

Table des matières

1 Introduction

Avec l'avènement des réseaux sociaux, Internet génère un nombre de données important. Chaque minute, il y a près de :

— 240 millions d'emails envoyés
— 2.46 millions de contenus partagés sur facebook
— 277 000 publiés sur twitter
— 216 000 nouveaux posts publiés sur Instagram

D'ici 2025, la quantité de données collectées chaque jour sera estimée à environ 463 exaoctets.

Notre objectif, en tant que data scientist de demain, est de développer des algorithmes permettant de traiter un grand nombre de données avec efficacité. Nous souhaitons optimiser au mieux nos algorithmes, afin que le temps d'exécution soit minimisé. Pour se faire, nous pouvons utiliser des méthodes de calcul parallèle tel que le Multiprocessing et multithreading. Au cours de ce travail, nous allons tout d'abord analyser de plus près ces deux techniques, enfin, nous finirons par les comparer à l'aide de données réelles.

2 Quelques définitions

Avant de débuter dans l'analyse de ces deux techniques, nous allons tout d'abord comprendre les bases : qu'est ce qu'un processus et des threads dans un ordinateur ?

2.1 Processus

Un processus, en informatique, est un programme en cours d'exécution par un ordinateur. Il est stocké dans la RAM, chaque processus a sa propre mémoire, ses données et autres ressources nécessaires à l'exécution du programme. Il peut y avoir plusieurs processus associés au même programme, par exemple, chaque onglet de votre Google Chrome est un processus en soi du programme chrome.exe résidant dans votre mémoire secondaire. Les processus étant stockés dans la mémoire vive, ils sont perdus lorsque le système est éteint. De plus, un seul processus peut être exécuté à tout moment par une seule unité centrale de traitement (CPU).

2.2 Thread

Il s'agit d'une entité qui réside dans un processus, chaque processus est lancé avec un seul thread appelé thread primaire. La spécificité du thread est qu'il laisse la possibilité à deux instances en train d'interpréter le même programme de s'exécuter en simultané au sein du même processeur.

Les threads sont similaires aux processus dans la mesure où les deux représentent l'exécution d'un ensemble d'instructions en langage machine à partir du processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle. Cependant, dans le cas où chaque processus a sa propre mémoire virtuelle, les threads du même processus partagent leur mémoire virtuelle. Cependant, tous les threads ont leur propre pile d'exécution.

Afin de mieux comprendre ce qui diffère les process des threads, nous pouvons voir ci-dessous un tableau de comparaison. Ci dessous, un exemple de notre ordinateur, avec le nombre de processus en cours (269) et de threads (3245).

	Processus	Thread
Définition	Un programme en cours d'exécution	Une petite partie d'un processus
Communication	La communication entre deux processus est coûteuse et limitée	La communication entre deux threads est moins coûteuse que celle du processus
Multitâche	Le multitâche basé sur les processus permet à un ordinateur d'exécuter deux ou plusieurs programmes simultanément	Le multitâche basé sur les threads permet à un programme unique d'exécuter deux threads ou plus simultanément
Espace d'adressage	Chaque processus a son espace d'adressage distinct	Tous les threads d'un processus partagent le même espace d'adressage que celui d'un processus
Tâches	Lourdes	Légères

TABLE 1 – Comparaison thread et process

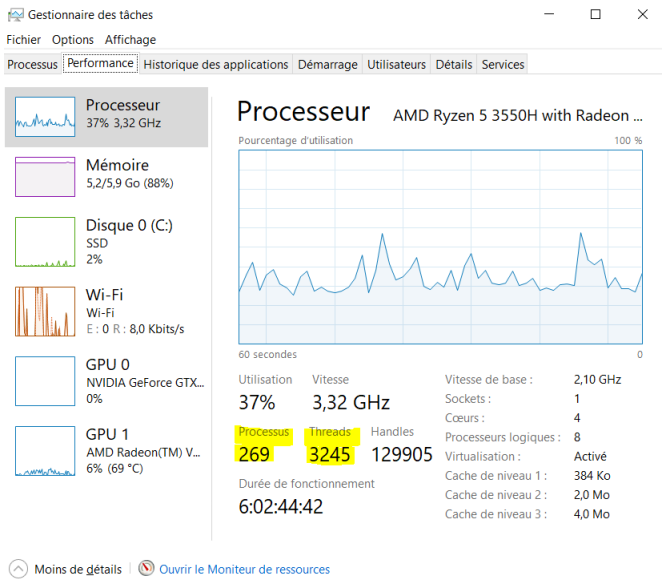

FIGURE 1 – Gestionnaire de tâches

Maintenant que nous avons posé les bases, nous pouvons rentrer dans le vif du sujet : le calcul parallèle.

3 Calcul parallèle

3.1 Multiprocessing

Le multiprocessing est un mode de fonctionnement dans lequel deux ou plusieurs processeurs d'un ordinateur traitent simultanément deux ou plusieurs parties différentes du même programme (Figure 2). Le multiprocessing est généralement effectué par deux microprocesseurs ou plus, chacun d'entre eux étant en fait une unité centrale de traitement (UC) sur une seule petite puce. Certains ordinateurs combinent des milliers de microprocesseurs de ce type pour interpréter et exécuter les instructions.

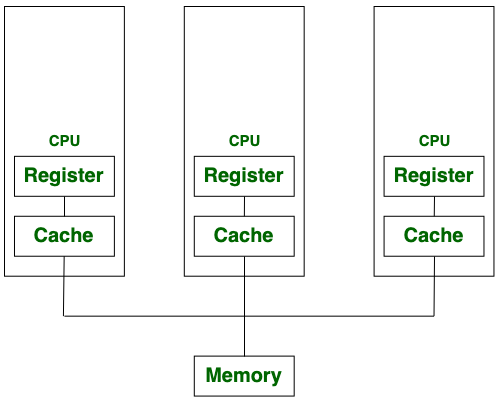

FIGURE 2 – Multiprocessing

3.2 Multithreading

Le multithreading est un système dans lequel plusieurs fils d'un processus sont créés pour augmenter la vitesse de calcul du système. Dans le multithreading, plusieurs threads d'un processus sont exécutés simultanément et la création de processus dans le multithreading se fait de manière économique (Figure 3).

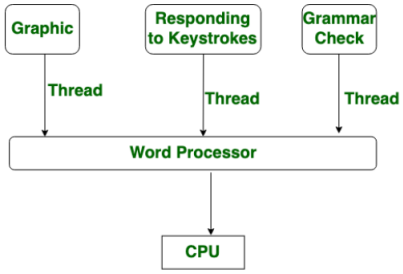

FIGURE 3 – Multithreading

Si nous soumettons des "tâches" à différents threads, ces tâches peuvent être considérées comme des "sous-tâches" d'un processus unique et ces threads auront généralement accès aux mêmes zones de mémoire (c'est-à-dire à la mémoire partagée). Cette approche peut facilement conduire à des conflits en cas de synchronisation incorrecte, par exemple, si des processus écrivent au même emplacement mémoire au même moment. Une approche plus sûre (bien qu'elle s'accompagne d'une surcharge supplémentaire due à la surcharge de communication entre des processus distincts) consiste à soumettre plusieurs processus à des emplacements mémoire complètement séparés (c'est-à-dire une mémoire distribuée) : Chaque processus s'exécutera de manière totalement indépendante les uns des autres.

4 Application et comparaison des deux méthodes

4.1 Méthode

4.1.1 Algorithmes

Nous allons comparer la performance et le temps de calcul avec multiprocessing, multithreading et en exécution séquentielle. Pour ce faire, nous avons effectué 4 algorithmes se trouvant dans 4 fichiers `.py` différents :

- `sleep.py`
- `calculation.py`
- `classification.py`

— `nlp.py`

Deux exemples simples Nous allons débuter par deux algorithmes simples.
— Le premier (`sleep.py`) est une fonction qui retarde l'exécution d'un nombre donné de secondes.
— Le second (`calculation.py`) est une fonction permettant de trouver le produit de tous les éléments d'une liste. Elle implique un grand nombre de calculs, et est donc gourmande en ressources CPU.

Exemple d'utilisation en apprentissage automatique Le but de cette expérimentation consistera à faire du tuning d'hyperparamètres de plusieurs modèles de classification. Le dataset contient des informations sur les réservations de 2 types d'hôtels, notamment la date de la réservation, la durée du séjour, le nombre d'adultes, d'enfants et/ou de bébés, le nombre de places de parking disponibles,... Avec ce dataset, nous souhaitons savoir si un client aura besoin ou non d'une place de parking pour son séjour : c'est donc un problème de classification binaire. A l'aide d'un `GridSearchCV`, nous testons plusieurs ensembles de paramètres pour quatre modèles différents : un k-NN, une régression logistique, un random forest et SVM. La validation croisée pour chacun de ces modèles s'effectuera donc séquentiellement (modèle après modèle) ou à l'aide de plusieurs processes ou threads.

Exemple d'utilisation en NLP L'expérimentation suivante consistera à traiter trois bases de données textuelles. Ces 3 bases de données contiennent respectivement des tweets politiques, des articles d'économie, et des résumés de films. Le but de cette expérimentation est de prétraiter ces 3 problèmes de NLP en retirant les caractères inutiles (hashtags, url, balises HTML), en tokenisant chaque texte et en appliquant un embedding type Word2Vec (résentation des mots dans un espace avec une forme de similarité entre eux, dans lesquels le sens des mots les rapproche dans cet espace, en terme de distances statistiques). Le prétraitement de ces 3 textes s'effectuera donc soit séquentiellement (texte après texte) ou à l'aide de plusieurs processes ou threads.

Exemple d'utilisation en Computer Vision Enfin, notre dernière expérimentation consistera à effectuer de la détection des visages. OpenCV est l'une des bibliothèques les plus populaires utilisées à cette fin. Nous allons travailler avec 187 images, chacune d'entre elles contenant une ou plusieurs personnes et l'objectif est de détecter la présence d'un visage dans une image et, si tel est le cas, d'entourer visage sur l'image.

4.1.2 Librairies

Nous pouvons utiliser les librairies suivantes :
— Multiprocessing : `multiprocessing` ou `concurrent.futures`
— Multithreading `threading` ou `multiprocessing.dummy` ou `concurrent.futures`

Peu importe la librairie utilisée, cela n'affectera pas les expérimentations. Nous allons donc utiliser pour le multiprocessing `multiprocessing` et pour le multithreading `threading`.

La bibliothèque de `multiprocessing` de Python offre deux façons d'implémenter le parallélisme basé sur les processus : `Process` et `Pool`.

Process Il est utilisé lorsque le parallélisme basé sur les fonctions est requis, où nous pouvons définir différentes fonctions avec les paramètres qu'elles reçoivent et exécuter en parallèle ces différentes fonctions qui effectuent des types de calculs totalement différents. Lorsque nous avons peu de données ou de fonctions et moins de tâches répétitives à effectuer. Cela met tout le processus dans la mémoire. Par conséquent, dans les tâches plus importantes, cela peut entraîner une perte de mémoire.

Pool Il offre un moyen pratique de paralléliser l'exécution d'une fonction sur plusieurs valeurs d'entrée, en distribuant les données d'entrée entre les processus, c'est-à-dire le parallélisme basé sur les données. Lorsque nous avons une quantité importante de données, il est préférable d'utiliser la classe Pool. Seuls les processus en cours d'exécution sont conservés en mémoire.

4.2 Résultats

TABLE 2 – Temps (secondes) d'exécution d'algorithmes avec différentes méthodes

	sleep	**calculation**	**classification**	**nlp**	**face_detection**
Séquentiel	6.0	78.3	73.8	80.9	7.5
Multithreading	3.0	83.6	60.7	77.3	10.3
Multiprocessing pool	3.8	26.3	47.0	56.8	4.4
Multiprocessing process	3.4	30.8	45.6	55.4	27.5

Analysons les résultats obtenues dans la table 2, comprenant le temps d'exécution de chacun des algorithmes cités précédemment avec les méthodes séquentielles, multiprocessing et multithreading.

sleep Lorsque nous utilisons le multithreading, le code ne prend qu'environ trois secondes pour terminer son exécution, contre six secondes lorsque nous le faisons en séquentiel. Contrairement à l'exécution en série, où sleep est appelé trois fois l'un après l'autre après l'achèvement de l'appel précédent, dans l'exécution multithreaded sleep est appelé trois fois simultanément, de sorte que le temps total pris pour compléter l'exécution du code est juste environ trois secondes (sleep(1) et sleep(2) auraient déjà été terminés au moment où sleep(3) est fait).

calculation Rappelons que cette fonction est gourmande en ressource CPU. Les résultats nous montrent que le multithreading est la moins bonne des options ici. En effet, en utilisant cette méthode, la durée d'exécution de l'algorithme est de 83 secondes, contre 78 en séquentiel. Tandis que le temps est divisé par 3 lorsque nous utilisons le multiprocessing pool. En effet, il est préférable d'utiliser cette classe lorsqu'il y a une quantité importante de calculs. L'utilisation du multithreading n'est pas bénéfique dans ce cas en raison de l'existence de la GIL en python. Toute opération / instruction est exécutée dans l'interpréteur. GIL garantit que l'interpréteur est détenu par un seul thread à un instant donné. Notre programme python avec plusieurs threads fonctionne dans un seul interpréteur. À tout instant particulier, cet interprète est détenu par un seul thread. Cela signifie que seul le thread qui contient l'interpréteur s'exécute à tout instant. En bref, sur python, le GIL s'assure qu'un seul de nos «threads» peut s'exécuter à la fois.

classification Pour la classification, le multiprocessing l'emporte encore une fois sur le séquentiel et le multithreading. Néanmoins, la différence de temps est moindre entre le multithreading et le multiprocessing comparé à l'algorithme précédent.

nlp Le temps d'exécution en séquentiel est de près de 81 secondes, contre 77 secondes en multithreading. Lorsqu'on a recours au multiprocessing, le temps diminue largement et passe à 57 secondes (pool) et 55 secondes (process). La différence de durée entre le multithreading et le multiprocessing s'explique encore une fois par l'existance du GIL qui permet que seulement un thread s'execute à la fois.

face_detection L'algorithme en cascade de Haar fonctionne sur un schéma de détection à fenêtre glissante à l'aide de plusieurs classifieurs consécutifs, ce qui représente beaucoup de travail pour le CPU. Il n'est donc pas surprenant de voir que le multiprocessing `Pool` est le plus performant. Paradoxalement, le multiprocessing en utilisant `Process` est le plus lent avec ses 27.5 secondes. En effet, `Process` permet le parallélisme sur les fonctions, c'est-à-dire plusieurs fonctions différentes mises en parallèle. Tandis que `Pool` offre un moyen pratique de paralléliser l'exécution d'une fonction sur plusieurs valeurs d'entrée, en distribuant les données d'entrée entre les processus. Avec 187 données d'entrées, il est donc préférable d'utiliser la classe `Pool`.

5 Conclusion

Grâce aux différentes implémentations d'algorithmes, nous avons pu analyser les différences entre multithreading et multiprocessing.

Dans la majorité des cas, lorsque les algorithmes étaient gourmands en ressources CPU (calculation, classification, nlp, face_detection), le multiprocessing permettait de largement diminuer la durée d'execution du programme. Le multithreading, quant-à-lui, n'était pas toujours bénéfique. Comme expliqué précédemment, GIL permet d'executer seulement un "thread" à la fois. Le sens littéral de multithreading n'existe donc pas vraiment sur python.

Ce projet nous a permis de nous rendre compte à quel point connaître sa machine était important pour faire de la data science. A présent, nous saurons optimiser nos programmes.

Références

[1] *C# et .NET : Version 1 à 4 - Gérard Leblanc - Google Livres*. URL : https://books.google.fr/books?id=TdRM4_Qo8aUC&pg=PA322&dq=%5C%22unit%5C%C3%5C%A9+de+traitement%5C%22+thread#v=onepage&q=%5C%22unit%5C%C3%5C%A9%5C%20de%5C%20traitement%5C%22%5C%20thread&f=false.

[2] Brendan FORTUNER. *Intro to Threads and Processes in Python*. Medium. 7 sept. 2017. URL : https://medium.com/@bfortuner/python-multithreading-vs-multiprocessing-73072ce5600b.

[3] *Les Chiffres des données générées chaque Minute sur les réseaux sociaux*. URL : https://www.arobasenet.com/2014/06/chiffres-chaque-minute-sur-reseaux-sociaux-996.html#gsc.tab=0.

[4] *multiprocessing — Parallélisme par processus — Documentation Python 3.9.5*. URL : https://docs.python.org/fr/3/library/multiprocessing.html.

[5] *Multiprocessing — computing — Britannica*. URL : https://www.britannica.com/technology/multiprocessing.

[6] *Multiprocessing vs. Threading in Python : What Every Data Scientist Needs to Know*. FloydHub Blog. 7 sept. 2019. URL : https://blog.floydhub.com/multiprocessing-vs-threading-in-python-what-every-data-scientist-needs-to-know/.

[7] *Techniques audiovisuelles et multimédias - 3e éd. : T2 : Systèmes micro ... - Gérard Laurent - Google Livres*. URL : https://books.google.fr/books?id=PQUGb_K40W0C&pg=PA54&dq=%5C%22unit%5C%C3%5C%A9+d%5C%E2%5C%80%5C%99ex%5C%C3%5C%A9cution%5C%22#v=onepage&q=%5C%22unit%5C%C3%5C%A9%5C%20d%5C%E2%5C%80%5C%99ex%5C%C3%5C%A9cution%5C%22&f=false.

[8] *Thread : définition simple et pratique*. URL : https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445320-thread-definition-simple-et-pratique/.