GIF-4104 - TP 5

Équipe 1 : Vincent Commin & Louis Leenart

1 Introduction

Pour ce dernier TP, nous avons eu à implémenter l'algorithme de *PageRank* en utilisant le moteur *Apache Spark*. L'algorithme consiste à associer un rang à chaque élément d'un graphe orienté. Dans notre cas, les noeuds du graphe représentent des URL, et les arêtes sont les références vers d'autres pages. Concernant l'implémentation, Spark support de nombreux langages et nous avons choisi Java pour sa simplicité d'utilisation ainsi que notre maîtrise du langage.

2 Notre approche

Pour implémenter cet algorithme, nous avons suivi la procédure suivante :

```
LECTURE fichier
CONSTRUCTION rdd A PARTIR de fichier

POUR iteration ALLANT DE 0 A 100 FAIRE
    CALCUL rang
FIN POUR

COLLECTE donnees

AFFICHE lien ET rang
```

Plus précisément, le calcul du rang pour chacun des liens se faire de la manière suivante :

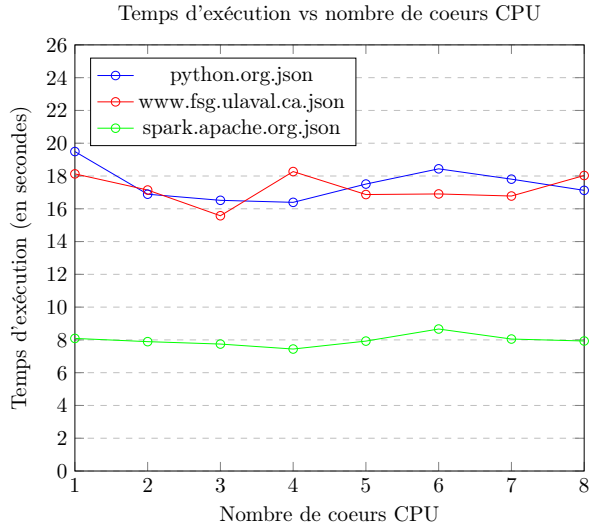
```
// un élément = id du lien, liste des id de lien qu'il référence
liens : LISTE COUPLE (id, LISTE id)
rangs : LISTE COUPLE (id, rang)
damping_factor : ENTIER

contributions : LISTE COUPLE (id, contribution)
contributions = join(liens, ranks).map(s -> {
    s.gauche.map(n -> {
        n, s.droit / len(s.gauche)
    })
})
rangs = contributions.reduce(+).map(s -> {
    s * damping_factor + 0.15
})
```

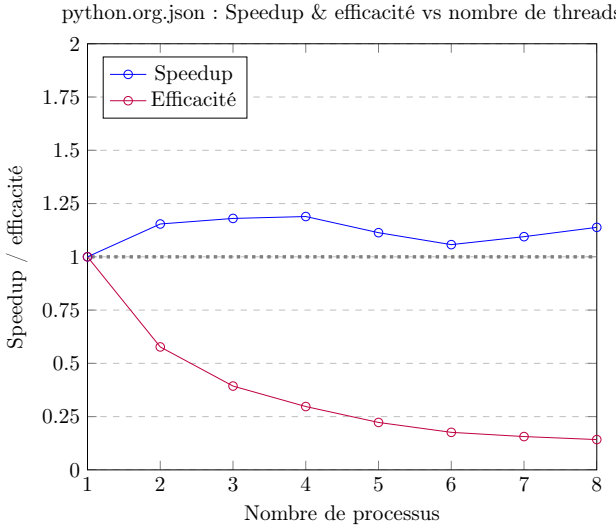
Après un nombre significatif d'intéractions, le contenu de 'rangs' contient les couples 'lien : rang', en sachant que plus le rang tends vers 0, moins il est considéré comme intéressant, et plus il est élevé, plus il est considéré comme pertinent.

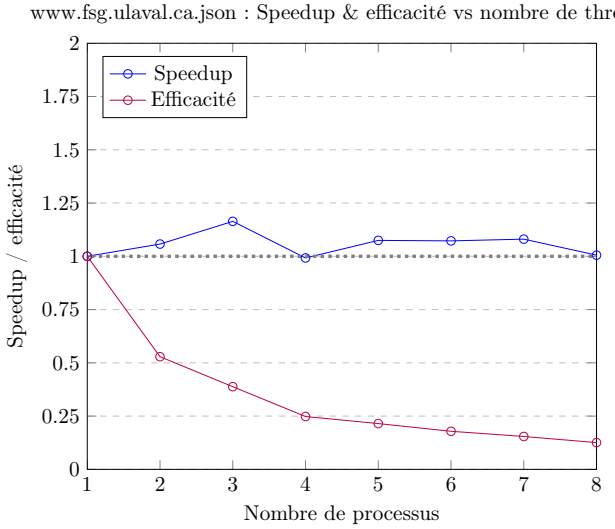
3 Machine utilisée pour les tests de performance

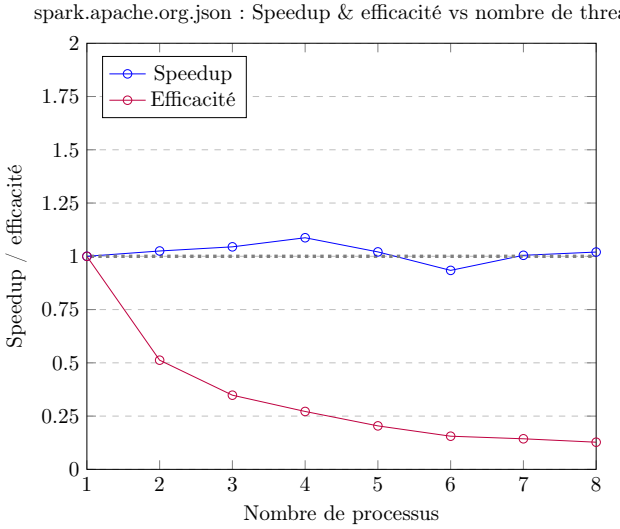
Modèle	intel i7-8550U
Architecture	x86_64
OS	Archlinux
Fréquence CPU	3.4GHz
Cœurs (physique / logique)	4 / 8
Ram	16 Go, 2400 MT/s
Java	OpenJDK 11.0.15
Spark Java	3.2.0

4 Résultats obtenus









5 Analyse

Pour mesurer les performances de notre algorithme et sa scalabilité avec le nombre de processus associés, nous avons utilisés les fichiers fournis, présents dans l'archive. Nous avons rencontré de nombreuses difficultés à paramétrer Spark, aussi bien pour le faire fonctionner de façon consistante, que pour spécifier le nombre exacte de CPU disponibles. Malgré tout, nous avons remarqué que Spark utilise plus de ressources qu'imposé. À noter aussi que Spark utilise plusieurs processus pour fonctionner, pour l'ordonnancement des processus et la gestion des ressources, utilisant alors du temps CPU difficile à mesurer.

D'après les graphiques ci-dessus, nous pouvons noter que le temps de calcul décroît légèrement avec l'ajout de cours CPU, cependant la diminution n'est pas significative, et parfois inexistante pour certains fichiers traités. Notamment pour le fichier "python.org.json", le speedup ne dépasse pas environ 1.25, ce qui est un résultat médiocre. L'efficacité est évidemment très faible aussi, presque égale au pire ($f(x) = \frac{1}{n}$) ce qui est particulièrement décevant. Ces résultats sont partagés pour les deux autres fichiers.

Après avoir essayé de nombreuses configurations, nous avons plusieurs théories pouvant expliquer ces résultats. Tout d'abord, la machine utilisée possède 4 cœurs physiques, or Spark demande plusieurs CPU pour pouvoir fonctionner (ordonnancement, gestion de l'accès aux ressources, etc). De ce fait, si Spark utilise déjà 2 voir 3 CPU, il n'y en a plus que 1 ou 2 cœurs (physiques) pour les calculs demandés. Cela signifie qu'il y a plus beaucoup de possibilité de paralléliser vu qu'il y a peu de cœurs CPU disponibles.

Pour un nombre de processus supérieur à 4, on commence à utiliser l'hyperthreading du CPU, donc des performances peu satisfaisantes. Enfin, novice quant à l'utilisation de Spark, il est absolument possible que nous n'avons pas utilisé les paramètres correctement.

6 Conclusion

Après avoir découvert et mis en place Spark, nous avons pu découvrir la mise en pratique d'une autre application de la programmation parallèle et distribuée, les architectures pour les données massives. Nous avons rencontrés des difficultés à paramétrer Spark, menant à des résultats de performance (speedup et efficacité) assez décevant. Malgré ces difficultés, nous avons tout de même réussi à faire fonctionner notre algorithme de PageRank, et avons pu commencer à expérimenter avec Spark. Concernant l'algorithme, étant donné sa simplicité, il n'est plus du tout utilisé de nos jours, et nous n'avons pas trouvé de perspectives d'amélioration le concernant. Basé sur l'analyse des liens d'un graphe, il brille par sa simplicité, cependant des alternatives modernes permettent d'obtenir des résultats concluant tout en profitant du parallélisme. Par exemple *Hyperlink-Induced Topic Search - HITS* et *Stochastic Approach for Link-Structure Analysis - SALSA* semble être de bons candidats pour ce genre d'application.