République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

# RAPPORT TP BIG DATA : CALCUL DE TF IDF

# AVEC SPARK PYTHON EN MAPREDUCE

## MASTER 2019/2020

**Binôme : (groupe 2)**

- **BRAHIM Yacine**
- **GACEM Abderaouf**

# Enoncé du problème :

Dans la recherche d'informations, TF – IDF ou TFIDF, en abrégé fréquence fréquence-document inverse, est une statistique numérique destinée à refléter l'importance d'un mot pour un document d'une collection ou d'un corpus. Il est souvent utilisé comme facteur de pondération. Dans les recherches de récupération d'information, de fouille de texte et de modélisation d'utilisateur. La valeur tf – idf augmente proportionnellement au nombre d'apparitions d'un mot dans le document et est compensée par le nombre de documents du corpus contenant le mot, ce qui permet de prendre en compte le fait que certains mots apparaissent plus souvent en général. Le tf – idf est l'un des schémas de pondération les plus populaires à l'heure actuelle, 83% des systèmes de recommandation basés sur du texte dans les bibliothèques numériques utilisent tf – idf. Dans ce travail, on calculera la TF et l'IDF en *MapReduce* dans l'environnement *Spark Python.*

*TF (t) = (Nombre de fois que le terme t apparaît dans un document) / (Nombre total de termes dans le document).*

*IDF (t) = log_e (Nombre total de documents / Nombre de documents contenant le terme t).*

# Technologie utilisée :

Spark est un moteur de traitement, parfois appelé Framework de traitement, est responsable de l'exécution des tâches de traitement de données. Apache Spark évolue dans un monde légèrement différent de celui de l'informatique ordinaire. Lorsque les ensembles de données deviennent trop volumineux ou que de nouvelles données arrivent trop rapidement, cela peut devenir trop difficile à gérer pour un seul ordinateur. C'est là que l'informatique distribuée entre en jeu. Au lieu d'essayer de traiter un grand ensemble de données ou d'exécuter des programmes très coûteux en calcul sur un ordinateur, ces tâches peuvent être réparties entre plusieurs ordinateurs qui communiquent entre eux pour produire un résultat. Cette technologie présente de sérieux avantages, mais l'attribution de tâches de traitement sur plusieurs ordinateurs présente ses propres défis et ne peut pas être structurée de la même manière qu'un traitement normal. Lorsque Spark indique qu'il s'agit de données distribuées, cela signifie qu'elles sont conçues pour traiter de très grands ensembles de données et pour les traiter sur un système informatique distribué. L'un des principaux avantages de Spark est sa flexibilité et le nombre de domaines d'application dont il dispose. Il prend en charge Scala, Python, Java, R et SQL. Il possède un module SQL dédié, il est capable de traiter les données en streaming en temps réel, et il est doté à la fois d'une bibliothèque d'apprentissage automatique et d'un moteur de calcul graphique. Toutes ces raisons expliquent

pourquoi Spark est devenu l'un des moteurs de traitement les plus populaires dans le domaine du Big Data.

# L’algorithme utilisé :

Pour le calcul de la TF et l’IDF, on considère deux collections de (clé, valeur) qu’on obtient de la façon suivante :

- L’information nécessaire pour calculer la TF d’un mot est le nombre de fois où ce dernier apparait dans l’ensemble des fichiers, pour cela on mappe chaque *mot* de tous les fichiers à *1* i.e. *(mot,1)* et on agrège par la somme, on aura ainsi *(mot, nombre_apparition)*. La TF d’un mot est obtenue ainsi par *(mot, nombre_apparition/ total_mots)*.
- L’information nécessaire pour calculer l’IDF d’un mot est le nombre de fichier dans lequel apparait le mot en question, pour l’avoir on mappe chaque *mot* de tous les fichiers à *1* mais en assurant l’unicité de cette pair par ficher, on agrège ensuite par la somme et on obtient ainsi une collection de *(mot, nombre_apparition_fichier)*. Le calcul de l’IDF s’obtient donc par *(mot, log(total_fichiers/ nombre_apparition_fichier))*

L’algorithme est conçu de façon à générer les deux collections en une seule passe, cela sert à garder de bonnes performances même avec l’augmentation de la taille de donnes à traiter.

# Test de scalabilité :

Spark garde les mêmes performances avec l’augmentation de nombre de fichiers en entrée, c’est d’ailleurs une de ses grandes contributions apportées au traitement de donnes. On peut voir sur la figure ci-dessous que l’évolution est linière et non pas exponentielle, il faut mentionner que l’échantillon n’est pas très représentatif mais reste que Spark est bien adapté aux données volumineuses.

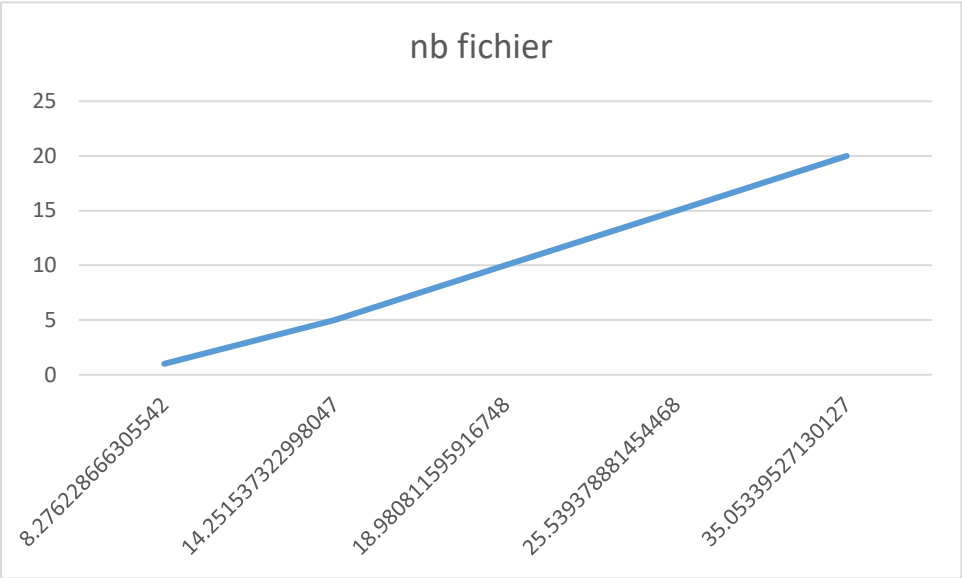


*Figure 1 temps d'exécution par nombre de fichier en entrée*

# Conclusion :

La réalisation de ce travail nous a permis de manipuler l'outil Spark python, de développer le raisonnement MapReduce et de voir l'évolution du temps d'exécution avec l'augmentation des donnes à traiter. On a pu établir que Spark est très adéquat au traitement de grande masse de donnes et que sa flexibilité de développer en plusieurs langage nous offre une aisance dans la programmation et la conception de l'algorithme.