REPUBLIQUE DU SENEGAL

Un peuple –un but –une foi

MINISTERE DE L'ENSEIGNEMENT SUPERIEURE DE LA RECHERCHE ET DE L'INNOVATION

DIRECTION GENERALE DE L'ENSEIGNEMENT SUPERIEUR

Université Gaston Berger

UFR Institut Polytechnique de Saint-Louis

Implémentation d'un programme MapReduce qui permet le comptage des mots d'un texte.

Présenté par :

Bassirou SAGNA

ING 3 Info-Telecom

Sous la direction de :

Dr. Djibril MBOUP

Plan :

Introduction :

Le traitement des données massives est devenu une nécessité incontournable dans de nombreux domaines, allant des moteurs de recherche à l'informatique, en passant par l'analyse financière et la recherche scientifique. Le paradigme MapReduce, introduit par Google, a révolutionné la manière dont nous abordons le traitement de grandes quantités de données en permettant leur traitement parallèle sur des clusters de machines. Ce rapport se concentre sur l'exercice "Word Count", qui vise à implémenter un programme MapReduce pour le comptage des mots dans un texte donné. Cet exercice nous servira de guide pour comprendre les concepts fondamentaux de MapReduce et pour apprendre à implémenter et exécuter des programmes MapReduce via notre machine virtuelle Vagrant.

L'une des objectifs de ce rapport est de nous permettre de se familiariser avec le paradigme MapReduce, qui se compose principalement de deux phases : la phase de mappage (Map) et la phase de réduction (Reduce). Chaque phase joue un rôle crucial dans le traitement des données distribuées. Ensuite à l'aide du langage Python par exemple qui a ses propres avantages et bibliothèques pour travailler avec Hadoop, le cadre de traitement de données le plus couramment utilisé pour exécuter des tâches MapReduce. Pour enfin finir avec le comptage des mots qui est une application classique de MapReduce. Cette tâche simple mais essentielle nous permettra de comprendre comment MapReduce peut être utilisé pour traiter des données textuelles massives. Cela inclut la lecture des données, la transformation des données en paires clé-valeur, et l'agrégation des résultats.

Pour couvrir les différents aspects de cet exercice de manière exhaustive, ce rapport est divisé en plusieurs sections :

- **Présentation du Paradigme MapReduce** : nous commencerons par un résumé détaillé du paradigme MapReduce, en expliquant les concepts clés, notamment la phase de mappage et la phase de réduction. Nous discuterons également de l'architecture et du flux de données dans un programme MapReduce typique.
- **Configuration de l'Environnement** : Une configuration appropriée de l'environnement sera essentielle pour le développement et l'exécution de programmes MapReduce. Cette section décrira les étapes nécessaires pour installer et configurer Hadoop sur une machine virtuelle Vagrant, y compris les configurations spécifiques pour chaque langage de programmation choisi à savoir le langage Python.

- **Implémentation du Programme Mapper et Reducer** : Dans cette section, nous détaillerons l'implémentation du programme Mapper et Reducer. Nous fournirons le code pour chaque fonction, en expliquant comment le Mapper transforme les données d'entrée en paires clé-valeur et comment le Reducer agrège ces paires pour produire le résultat final.
- **Exécution et Résultats** : Une fois le programme implémenté, nous le déploierons sur un cluster Hadoop et présenterons le processus d'exécution. Cette section inclura également une analyse des résultats obtenus, en montrant comment les mots ont été comptés et agrégés à partir du texte d'entrée.
- **Conclusion** : Enfin, nous résumerons les résultats de l'exercice et les leçons apprises. Nous discuterons des défis rencontrés et des solutions mises en œuvre, ainsi que des améliorations potentielles et des applications futures de cette technique.

Présentation du Paradigme MapReduce :

Le paradigme MapReduce est une technologie fondamentale développée par Google pour le traitement et la génération de grands ensembles de données. MapReduce se distingue par sa capacité à diviser et à conquérir les tâches de traitement des données en les décomposant en étapes plus simples et en les exécutant en parallèle sur un grand nombre de machines. Ce paradigme repose sur deux fonctions principales : la fonction de mappage (Map) et la fonction de réduction (Reduce).

- Phase de Mappage (Map) : dans le cadre d'un programme de comptage de mots, la fonction Map lit un morceau de texte, sépare le texte en mots individuels, et émet une paire clé-valeur pour chaque mot, où la clé est le mot et la valeur est le nombre 1.
- Phase de Réduction (Reduce) : avec l'exemple du comptage de mots, la fonction Reduce prend toutes les paires clé-valeur pour chaque mot, les regroupe, et somme les valeurs pour obtenir le nombre total de fois que chaque mot apparaît dans le texte.

Le paradigme MapReduce, avec ses phases de mappage et de réduction, offre une approche robuste et scalable pour le traitement de grandes quantités de données. En décomposant les tâches en étapes simples et en les exécutant en parallèle sur un cluster de machines, MapReduce permet de traiter des données à une échelle et à une vitesse qui seraient impossibles à réaliser avec des méthodes traditionnelles. Dans ce rapport, nous allons explorer ces concepts en détail et illustrer leur application à travers l'implémentation d'un programme de comptage de mots en utilisant Hadoop et l'un des langages de programmation choisis.

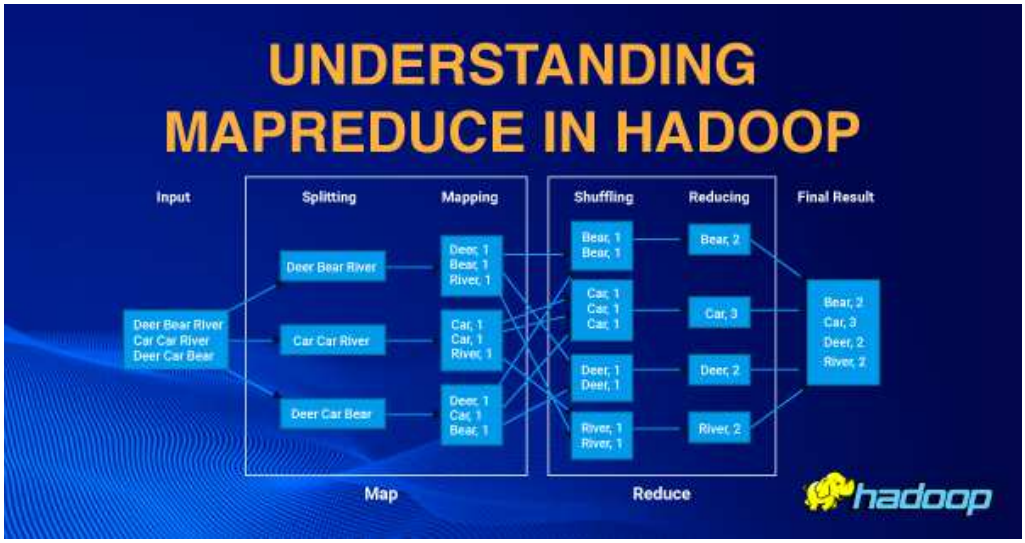

Figure illustratif de Mapreduce avec Hadoop pour le Mappage et la Reduction

Configuration de l'Environnement :

S'agissant de la configuration de l'environnement nous allons à l'aide du lien git partager dans la cour https://github.com/sopeKhadim/hadoopVagrant.git cloner le dossier à l'aide de la commande **git clone** comme le montre la capture d'écran ci-dessous :

```
lenovo@B-SAGNA MINGW64 ~/Desktop/Documents
$ git clone https://github.com/sopeKhadim/hadoopVagrant.git
Cloning into 'hadoopVagrant'...
remote: Enumerating objects: 49, done.
remote: Counting objects: 100% (49/49), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 49 (delta 9), reused 30 (delta 5), pack-reused 0
Receiving objects: 100% (49/49), 1.72 MiB | 229.00 KiB/s, done.
Resolving deltas: 100% (9/9), done.

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents
$
```

Une fois le dossier cloner nous allons de ce fait passer aux étapes suivantes des configurations pour pouvoir exécuter normalement les commandes venir nous allons en premier lieu entrer dans notre dossier sur le : **C:\Users\lenovo\Desktop\Documents\hadoopVagrant\.vagrant\machines\default\virtualbox** dans le fichier **creator_uid** mettre le nombre qui se trouve initialement à **1000** le mettre à **0** une fois cette étape terminer nous pouvons de ce fait commencer à importer la machine virtuelle pour se faire nous allons modifier le fichier **VagrantFile** avant de modifier le fichier nous devons au préalable créer un dossier partager ou pour permettre à notre machine virtuelle de voir les dossiers de notre machine hôte. Pour les besoins de notre travail nous allons créer un dans notre dossier racine un nouveau dossier du nom de **hadoop** comme le montre la capture ci-dessous :

```
lenovo@B-SAGNA MINGW64 ~/Desktop/Documents
$ cd hadoopVagrant/

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ ls
README.md  Vagrantfile  scripts/

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ mkdir hadoop

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ ls
README.md  Vagrantfile  hadoop/  scripts/
```

Notons bien que c'est dans ce nouveau dossier créer que nous allons mettre le code du mappage et le code de la réduction ainsi que le fichier texte pour faciliter ainsi les dossiers de pouvoir

s'exécuter. Après la création de notre dossier partager nous allons modifier le fichier VagrantFile en fonction des paramètres de nos configurations comme suit :

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "SopeKhadim/hadoopVM"
  config.vm.box_version = "2.0"

  # Configuration du réseau public bridgé
  config.vm.network :public_network, :bridge => "Thiapathioly"

  # Configuration du dossier partagé
  config.vm.synced_folder "C:/Users/lenovo/Desktop/hadoopVagrant", "C:/Users/lenovo/Desktop/hadoopVagrant/hadoop"

  # Configuration des ressources de la machine virtuelle VirtualBox
  config.vm.provider "virtualbox" do |vb|
    vb.cpus = "4"
    vb.memory = "6096"
  end
end
```

Après avoir mis les différents paramètres de notre machine avant de le lancer nous allons dans notre dossier git bash et créer le dossier partagé afin que l'on puisse respecter les paramètres entrés en amont dans les configurations comme suit :

```
lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ ls
README.md  Vagrantfile  scripts/

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ mkdir hadoop

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ ls
README.md  Vagrantfile  hadoop/  scripts/

lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$
```

Une fois la creation du dossier partager créer nous allons ensuite passer au lancement de notre machine avec la commande **vagrant up** qui est utilisée pour démarrer et provisionner l'environnement spécifié par notre fichier Vagrantfile en :

- Vagrant lit le fichier **Vagrantfile** pour déterminer la configuration de la machine virtuelle à créer.
- Si la machine virtuelle spécifiée n'existe pas déjà, Vagrant télécharge et installe l'image de base indiquée dans le Vagrantfile, en d'autre mots Vagrant crée et configure une nouvelle machine virtuelle en utilisant cette image de base.

- Vagrant configure la machine virtuelle en fonction des directives spécifiées dans le Vagrantfile, comme la configuration du réseau, le montage des dossiers partagés, l'installation de logiciels supplémentaires, etc.
- Il permet aussi de démarrer la machine virtuelle.

```
lenovo@B-SAGNA MINGW64 ~/Desktop/Documents/hadoopVagrant (main)
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'SopeKhadim/hadoopVM' could not be found. Attempting to find and install...
    default: Box Provider: virtualbox
    default: Box Version: 2.0
==> default: Loading metadata for box 'SopeKhadim/hadoopVM'
    default: URL: https://vagrantcloud.com/api/v2/vagrant/SopeKhadim/hadoopVM
==> default: Adding box 'SopeKhadim/hadoopVM' (v2.0) for provider: virtualbox
    default: Downloading: https://vagrantcloud.com/SopeKhadim/boxes/hadoopVM/versions/2.0/providers/virtualbox/unknown/vagrant.box
    default:
==> default: Successfully added box 'SopeKhadim/hadoopVM' (v2.0) for 'virtualbox'!
==> default: Importing base box 'SopeKhadim/hadoopVM'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'SopeKhadim/hadoopVM' version '2.0' is up to date...
==> default: Setting the name of the VM: hadoopVagrant_default_1721675678205_63592
==> default: Fixed port collision for 22 => 2222. Now on port 2200.
```

Une fois avoir fini le téléchargement de la machine nous allons essayer de nous connecter à cette machine via la commande **vagrant ssh :**

```
C:\Users\lenovo\Desktop\Documents\hadoopVagrant>vagrant ssh
Last login: Sun Nov 28 11:04:44 2021 from 10.0.2.2
[vagrant@10 ~]$
```

Cette capture nous montre que nous somme bien connecter à notre machine virtuelle vagrant.

Implémentation du Programme Mapper et Reducer :

Dans cette partie nous allons directement nous lancer dans la configuration de nos deux programme en allant dans notre dossier partager pour ensuite y créer les fichiers nécessaires à la réalisation de notre travail.

La capture ci-dessous nous montre comment nous allons nous placer dans notre fichier partager et créer les fichiers ici nous allons utiliser le langage python pour la réalisation de notre travail :

Une fois placer dans notre fichier partager nous allons créer le fichier **mapper.py** qui est est un script Python qui est typiquement utilisé dans un contexte de traitement de données en utilisant le modèle de programmation MapReduce qui lui aussi est un modèle de programmation pour le traitement et la génération de grands ensembles de données en parallèle sur un cluster.

L'utilisation avec Hadoop ou d'autres frameworks MapReduce pour effectuer des tâches telles que le comptage de mots dans de grands ensembles de données. Le script mapper génère des paires clé-valeur pour chaque mot, et ces paires sont ensuite triées et regroupées par clé avant d'être transmises à un reducer, qui somme les valeurs pour chaque clé pour obtenir le nombre total de chaque mot.

Configuration de notre fichier **mapper.py :**

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys

# Lire l'entrée ligne par ligne
for line in sys.stdin:
    # Supprimer les espaces blancs en début et fin de ligne
    line = line.strip()
    # Diviser la ligne en mots
    words = line.split()
    # Émettre la paire clé-valeur (mot, 1) pour chaque mot
    for word in words:
        print('%s\t%s'%(word,1))
```

Une fois avoir fini avec le fichier mapper nous allons passer à notre fichier **reduce.py** qui à l'instar de mapper est une autre composante du modèle de programmation MapReduce, complémentaire au script **mapper.py**. Alors que mapper.py émet des paires clé-valeur intermédiaires, reducer.py prend ces paires et produit les résultats finaux en agrégant les valeurs pour chaque clé.

Le script reducer.py est souvent utilisé dans les systèmes de traitement de données distribuées comme Hadoop, où les mappers et reducers sont déployés sur un cluster de machines pour traiter de grands ensembles de données en parallèle. Le reducer agrège les résultats des mappers, ce qui permet de calculer des totaux, des moyennes, des maximas, des minimas, etc., pour chaque clé.

Configuration de notre fichier **reducer.py :**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys

prev_word = None
prev_count = 0

# Lire l'entrée ligne par ligne
for line in sys.stdin:
    # Supprimer les espaces blancs en début et fin de ligne
    line = line.strip()
    # Diviser la ligne en mot et compte
    word, count = line.split('\t')
    count = int(count)

    # Si le mot actuel est le même que le précédent, augmenter le compte
    if prev_word == word:
        prev_count += count
    else:
        # Sinon, afficher le résultat pour le mot précédent
        if prev_word:
            print('%s\t%s' % (prev_word, prev_count))
        prev_count = count
        prev_word = word

# Afficher le dernier mot
if prev_word == word:
    print('%s\t%s' % (prev_word, prev_count))
```

Exécution et Résultats :

Une fois les deux fichiers créent et copier à dans notre fichier partager à savoir mapper et reduce nous allons aussi y placer le fichier à que nous allons effectuer le compte de savoir ancien_figaro.txt comme le montre la capture suivante :

```
[vagrant@10 /]$ ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  vagrant  var
[vagrant@10 /]$ cd vagrant/
[vagrant@10 vagrant]$ cd hadoop/
[vagrant@10 hadoop]$ ls
ancien_figaro.txt  mapper.py  reducer.py
[vagrant@10 hadoop]$
```

Une fois que nous avons vu les dossiers nous allons commencer à voir en premier si le mapper marche avec la commande **cat ancien_figaro.txt | python mapper.py** qui nous affiche le résultat ci-dessous :

```
conférences     1
de      1
Pilnitz;        1
il      1
est     1
campé   1
sur     1
les     1
bords   1
du      1
Rhin,   1
et      1
il      1
attend  1
que     1
M.      1
de      1
Brunswick       1
ait     1
mis     1
à       1
la      1
raison  1
les     1
Parisiens       1
```

Cette capture nous montre bien que le compte des mots s'est bien fait nous allons ensuite essayer avec le reducer pour voir si le programme marche avec la commande n'oublions pas que les deux commandes sont liés ce qui donnent : **cat ancien_figaro.txt | python mapper.py | sort | python reducer.py**

```
XIV,    2
XIV!    1
XIV.    3
XIV     7
XV.     1
XV      1
XVI,    1
XVI     1
XVIII,  3
XVIII.) 1
XVIII   2
y       148
years,  1
```

Cette capture nous permet de vois que notre comptage des mots c'est bien passée et que le mits sont bien comptés.

Conclusion :

Dans cette étude, nous avons exploré les étapes fondamentales du modèle de programmation MapReduce à travers l'exemple d'un programme de comptage de mots. Ce modèle est largement utilisé pour le traitement et l'analyse de grands ensembles de données distribuées, notamment dans des environnements de calcul parallèle comme Hadoop.

Le script `mapper.py` a été conçu pour lire des lignes de texte en entrée, diviser chaque ligne en mots, et émettre une paire clé-valeur pour chaque mot. Chaque paire est composée du mot comme clé et de `1` comme valeur. Ce processus de mappage permet de transformer les données brutes en paires clé-valeur intermédiaires qui facilitent l'agrégation ultérieure.

Le script `reducer.py` prend en entrée les paires clé-valeur produites par le mapper. Il agrège les valeurs pour chaque clé (mot) et émet le total des occurrences pour chaque mot. Ce processus de réduction permet de condenser les données intermédiaires en résultats finaux significatifs.

Les concepts illustrés par les scripts `mapper.py` et `reducer.py` sont appliqués dans divers domaines, tels que l'analyse de journaux, le traitement de données textuelles, et le calcul statistique à grande échelle. Les environnements de traitement de données distribuées comme Hadoop exploitent ces principes pour réaliser des tâches complexes telles que le comptage de mots, la classification de texte, et l'analyse de tendances.

Cette étude a mis en lumière les étapes essentielles du traitement de données avec MapReduce, en illustrant comment transformer des données brutes en résultats agrégés significatifs à l'aide de scripts de mappage et de réduction. En abordant les défis liés à la gestion des encodages et à la validation des données, nous avons démontré l'importance de la rigueur dans le développement de scripts de traitement de données. Cette approche est fondamentale pour tirer parti des puissantes capacités de traitement parallèle offertes par les technologies modernes, et elle constitue une base solide pour des analyses de données avancées et à grande échelle.