Questions théoriques sur Hadoop (25 questions)

1. **Qu'est-ce que Hadoop, et quels sont ses composants principaux ?**
   Réponse : Hadoop est un framework open source permettant de stocker et traiter de grandes quantités de données de manière distribuée. Ses composants principaux sont HDFS (stockage) et MapReduce (traitement).
2. **Quel problème Hadoop résout-il principalement ?**
   Réponse : Hadoop résout les problèmes liés à la gestion et au traitement de grands volumes de données non structurées de manière efficace et à faible coût.
3. **Qu'est-ce que HDFS, et quelle est sa fonction ?**
   Réponse : HDFS (Hadoop Distributed File System) est un système de fichiers distribué qui permet de stocker des données volumineuses sur plusieurs nœuds de manière fiable.
4. **Qu'est-ce que le traitement parallèle dans Hadoop, et à quoi sert-il ?**
   Réponse : Le traitement parallèle consiste à diviser une tâche en sous-tâches qui sont exécutées simultanément sur plusieurs nœuds. Cela accélère le traitement des données massives.
5. **Quelles sont les étapes principales d'un job MapReduce ?**
   Réponse : Les étapes principales sont :
   - **Map** : Transformation des données d'entrée en paires clé-valeur.
   - **Shuffle & Sort** : Regroupement et tri des données par clé.
   - **Reduce** : Agrégation ou traitement final des données par clé.
6. **Pourquoi HDFS divise-t-il les fichiers en blocs ?**
   Réponse : Pour répartir les fichiers volumineux sur plusieurs nœuds et permettre un traitement parallèle.
7. **Quelle est la taille par défaut d'un bloc HDFS ?**
   Réponse : 128 Mo (configurable selon la version de Hadoop).
8. **Quels sont les avantages de la redondance dans HDFS ?**
   Réponse : La redondance garantit la disponibilité et la fiabilité des données en cas de défaillance d'un nœud.
9. **Quelle est la fonction du NameNode dans HDFS ?**
   Réponse : Le NameNode gère les métadonnées des fichiers (emplacement des blocs, structure des répertoires, etc.).
10. **Qu'est-ce qu'un DataNode dans HDFS ?**
    Réponse : Les DataNodes stockent les blocs de données et exécutent les instructions du NameNode.
11. **Quelles sont les opérations de lecture et d'écriture sur HDFS ?**
    Réponse :
    - **Lecture** : Le client demande les métadonnées au NameNode, puis lit les blocs directement depuis les DataNodes.
    - **Écriture** : Les données sont d'abord envoyées au NameNode pour les métadonnées, puis écrites sur les DataNodes en cascade.
12. **Comment HDFS gère-t-il la tolérance aux pannes ?**
    Réponse : Grâce à la réplication des blocs sur plusieurs DataNodes.
13. **Quels sont les trois états d'un job MapReduce ?**
    Réponse : Soumis, En cours d'exécution, Terminé.

14. **Qu’est-ce que le combiner dans MapReduce ?**
    Réponse : Une fonction facultative qui effectue une réduction locale des données sur les nœuds map avant l’étape shuffle, pour réduire le trafic réseau.
15. **Quelles sont les limites de MapReduce ?**
    Réponse : Latence élevée pour les petites tâches, complexité de programmation, pas adapté aux calculs interactifs ou en temps réel.
16. **Qu’est-ce que le job tracker dans Hadoop 1.x ?**
    Réponse : Il coordonne les tâches MapReduce en assignant des tâches aux nœuds et en surveillant leur exécution.
17. **Quelle est la différence entre Hadoop 1.x et Hadoop 2.x ?**
    Réponse : Hadoop 2.x introduit YARN (Yet Another Resource Negotiator) pour une meilleure gestion des ressources et le support de multiples frameworks au-delà de MapReduce.
18. **Quels sont les cas d'utilisation courants de Hadoop ?**
    Réponse : Analyse de logs, moteur de recommandation, traitement de données non structurées, calcul scientifique.
19. **Quelle est la commande pour télécharger un fichier sur HDFS ?**
    Réponse : `hdfs dfs -put <source> <destination>`.
20. **Comment vérifier l'espace utilisé sur HDFS ?**
    Réponse : `hdfs dfs -du -h /`.
21. **Qu’est-ce que la co-localisation des données dans Hadoop ?**
    Réponse : Le traitement des données se fait sur les nœuds où elles sont stockées, réduisant ainsi le transfert réseau.
22. **Qu’est-ce qu’un Partitioner dans MapReduce ?**
    Réponse : Un composant qui décide à quel Reducer une paire clé-valeur doit être envoyée.
23. **Pourquoi le shuffle & sort est-il une étape critique dans MapReduce ?**
    Réponse : Il regroupe les données par clé avant la phase reduce, garantissant que chaque Reducer traite une seule clé.
24. **Qu'est-ce qu'une application WordCount dans MapReduce ?**
    Réponse : Un exemple classique où le programme compte le nombre d'occurrences de chaque mot dans un texte.
25. **Quels outils peuvent être utilisés avec Hadoop pour des tâches Big Data ?**
    Réponse : Hive, Pig, Spark, Flume, Sqoop.

Questions pratiques sur Hadoop (25 questions)

26. **Écrire le pseudocode pour une tâche WordCount en MapReduce.**
    Réponse :
- Mapper : Émettre (mot, 1) pour chaque mot dans une ligne.
- Reducer : Additionner les valeurs pour chaque mot.

27. **Comment configurer un réplica de 2 pour un fichier HDFS ?**
    Réponse : Utiliser la commande `hdfs dfs -setrep -w 2 <file>`.
28. **Donner une commande pour lister les fichiers dans un répertoire HDFS.**
    Réponse : `hdfs dfs -ls <path>`.
29. **Quelles sont les étapes pour exécuter un job MapReduce ?**
    Réponse :
    - Compiler le programme MapReduce.
    - Charger les données d'entrée dans HDFS.
    - Soumettre le job via `hadoop jar`.
30. **Comment vérifier les logs d'un job MapReduce en cours ?**
    Réponse : Utiliser l'interface web du ResourceManager ou des commandes `yarn logs`.
31. **Quelle commande supprime un fichier dans HDFS ?**
    Réponse : `hdfs dfs -rm <file>`.
32. **Quelle commande récupère un fichier depuis HDFS ?**
    Réponse : `hdfs dfs -get <source> <destination>`.
33. **Qu'est-ce qu'un fichier JAR dans Hadoop ?**
    Réponse : Un fichier contenant le code Java exécutable d'un job MapReduce.
34. **Comment vérifier l'état d'un job MapReduce ?**
    Réponse : Utiliser `yarn application -status <applicationId>`.
35. **Donner une commande pour compter le nombre de lignes dans un fichier sur HDFS.**
    Réponse : `hdfs dfs -cat <file> | wc -l`.
36. **Expliquer la différence entre InputSplit et Block dans HDFS.**
    Réponse : Les blocs sont des segments physiques, tandis qu'InputSplit est une unité logique pour le traitement.
37. **Donner un exemple d'application de la fonction Combiner.**
    Réponse : Somme intermédiaire des valeurs dans une tâche WordCount.
38. **Quel format de fichier est recommandé pour MapReduce ?**
    Réponse : Avro ou Parquet, car ils sont compacts et optimisés.
39. **Comment configurer Hadoop sur un cluster ?**
    Réponse : Configurer les fichiers `core-site.xml`, `hdfs-site.xml` et `mapred-site.xml`.
40. **Comment exécuter un programme WordCount en Hadoop ?**
    Réponse :
    - Charger le fichier d'entrée sur HDFS.
    - Lancer le job avec `hadoop jar`.
    - Récupérer la sortie de HDFS.

41. **Comment exécuter un job Hadoop en mode pseudo-distribué ?**
    Réponse : Configurer Hadoop avec un seul nœud agissant comme NameNode et DataNode.
42. **Comment éviter un goulet d'étranglement dans Shuffle & Sort ?**
    Réponse : Augmenter le nombre de partitions ou configurer les paramètres mémoire adéquats.
43. **Quelles sont les extensions de fichiers par défaut utilisées par MapReduce ?**
    Réponse : `.crc`, `.class`, `.job`.
44. **Comment configurer la réplication par défaut dans HDFS ?**
    Réponse : Modifier le paramètre `dfs.replication` dans `hdfs-site.xml`.
45. **Quel est le rôle de YARN dans Hadoop ?**
    Réponse : Gestion des ressources et planification des tâches sur un cluster.
46. **Comment tester un job MapReduce en local ?**
    Réponse : Utiliser `LocalJobRunner` pour exécuter le job sur un nœud unique.
47. **Donner un cas d'utilisation de MapReduce autre que WordCount.**
    Réponse : Analyse de logs pour détecter des erreurs.
48. **Quelle commande permet de vérifier si HDFS est en cours d'exécution ?**
    Réponse : `jps` pour vérifier si `NameNode` et `DataNode` sont actifs.
49. **Comment nettoyer les sorties précédentes avant d'exécuter un job MapReduce ?**
    Réponse : Supprimer le répertoire de sortie avec `hdfs dfs -rm -r <output_path>`.
50. **Quels sont les outils nécessaires pour configurer un environnement Hadoop ?**
    Réponse : Java JDK, SSH, Hadoop binaries.

QCM

51. **L'unique changement de Hadoop v1 à Hadoop v2 était la séparation de la gestion des ressources et des traitements des jobs.**
    Réponse : Faux
    - La séparation de la gestion des ressources (YARN) et des traitements des jobs est l'un des principaux changements, mais pas le seul. Hadoop v2 introduit également :
        i. **YARN (Yet Another Resource Negotiator)** pour gérer les ressources indépendamment des applications.
        ii. **Support pour des applications non-MapReduce**, permettant d'exécuter d'autres types de workloads.
        iii. **Tolérance aux pannes améliorée** et **meilleure scalabilité** grâce à un NameNode secondaire actif (HA - High Availability).

52. **Lequel de ces composants n'existe pas dans la distribution Cloudera 4.7 ?**

- **java**
- **pig**
- **spark**

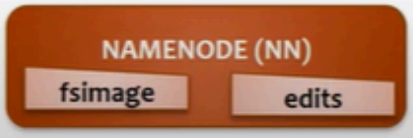
- **hue**
  Réponse : spark
- Cloudera 4.7 ne supporte pas **Spark**, car il est apparu dans les versions ultérieures. Les autres composants (Java, Pig, Hue) étaient disponibles.

HDFS READ AND WRITE

read :
1-le client **communique** avec le name node pour obtenir les **metadata** et les **emplacements** des data nodes contenant des block.
2-une fois le client reçoit les emplacements interager avec les data nodes
3-le client commence a **lire les donnees en parallele** a partir des datanodes en fonction des infors recue du namenode
4-les **donnees circulerant directement du data node vers le client**

Write :
1-le client **communique** avec le namenode pour obtenir les méta data
2-le name node **vérifier** si le fichier est **dispo** ou non , ainsi que si le client est **autorisé** ou non
3-le NameNode répond avec certain nombre de **bloc**,leur **emplacement**, leur **réplication** et d'autre détails sur la bases des infos du name node
4-le client interagit directement avec le data node

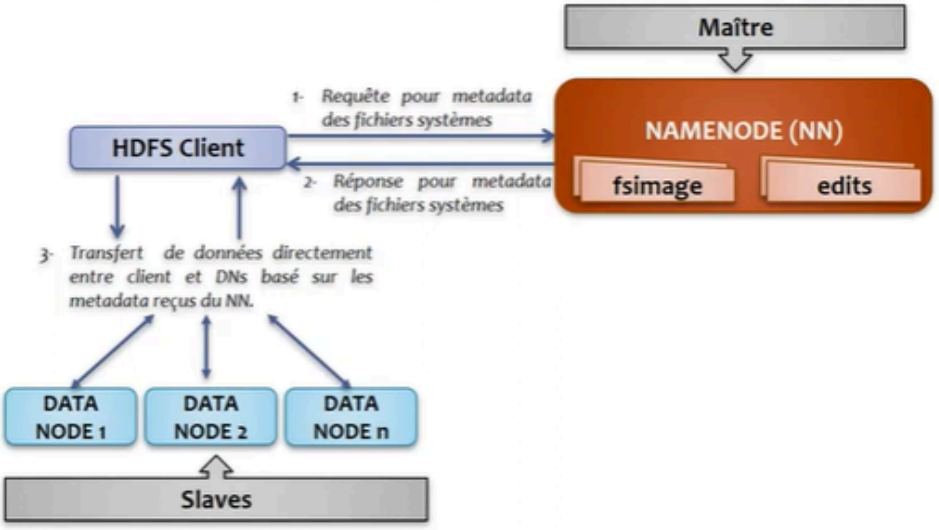
- **NAMENODE**
  - NameNode s'exécute sur la machine « Master ».
  - Il se compose de fichiers et de répertoires et ne stocke pas les données réelles.
- **LES FONCTIONS DU NAMENODE**
  - Garde en **RAM** les **métadonnées** des fichiers:
    - la taille des fichiers, les permissions
    - le nombre de blocs de données,
    - L'emplacement des blocs de données, sur quel rack et quel datanode,...
  - Responsable de la surveillance et de la gestion des DataNodes.
  - Responsable du « namespace » du système de fichiers.
  - Capture toutes les modifications apportées aux métadonnées, telles que la suppression, la création et le changement de nom du fichier dans les journaux d'édition.
  - Reçoit chaque 3 secondes des « signaux de pulsation » depuis les DataNodes.
  - Exécute les opérations du système de fichiers ouvrir, fermer,.. des fichiers et des répertoires.



- **Fsimage :** C'est une copie des métadonnées du système de fichiers.
- **Edits :** Ce fichier stocke les modifications apportées aux méta-informations

DATANODES

- DataNode s'exécute sur la machine « Slave ».
- Un cluster HDFS typique a plusieurs DataNodes
- Les DataNodes stockent les blocs de données (les données réelles).
- DN effectue la création, la réplication et la suppression des blocs conformément aux instructions de NN.
- Toutes les 3 secondes, par défaut, il envoie une pulsation à NameNode pour signaler l'intégrité du système HDFS.



code java word count :

mapper

```
public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,Int
Writable>{
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
   public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
       Reporter reporter) throws IOException{
      String line = value.toString();
      StringTokenizer  tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()){
         word.set(tokenizer.nextToken());
         output.collect(word, one);
      }
   }

}
```

reducer 👍

```
public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
 Reporter reporter) throws IOException {
int sum=0;
while (values.hasNext()) {
sum+=values.next().get();
}
output.collect(key,new IntWritable(sum));
}
}
```

runner :

```
public class WC_Runner {
   public static void main(String[] args) throws IOException{
      JobConf conf = new JobConf(WC_Runner.class);
      conf.setJobName("WordCount");
      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);
      conf.setMapperClass(WC_Mapper.class);
      conf.setCombinerClass(WC_Reducer.class);
      conf.setReducerClass(WC_Reducer.class);
      conf.setInputFormat(TextInputFormat.class);
      conf.setOutputFormat(TextOutputFormat.class);
      FileInputFormat.setInputPaths(conf,new Path(args[0]));
      FileOutputFormat.setOutputPath(conf,new Path(args[1]));
      JobClient.runJob(conf);
   }
}
```

CHar count
mapper

```
public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{
   public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
       Reporter reporter) throws IOException{
      String line = value.toString();
      String  tokenizer[] = line.split("");
      for(String SingleChar : tokenizer)
      {
         Text charKey = new Text(SingleChar);
         IntWritable One = new IntWritable(1);
         output.collect(charKey, One);
      }
   }

}
```

reducer

```
public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
 Reporter reporter) throws IOException {
int sum=0;
while (values.hasNext()) {
sum+=values.next().get();
}
output.collect(key,new IntWritable(sum));
}
}
```

runner 👍

```
public class WC_Runner {
  public static void main(String[] args) throws IOException{
    JobConf conf = new JobConf(WC_Runner.class);
    conf.setJobName("CharCount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WC_Mapper.class);
    conf.setCombinerClass(WC_Reducer.class);
    conf.setReducerClass(WC_Reducer.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf,new Path(args[0]));
    FileOutputFormat.setOutputPath(conf,new Path(args[1]));
    JobClient.runJob(conf);
  }
}
```

Execution With java

1) **create file:**

wordcount :

-WordCountDriver.java

-WordCountMapper.java

-WordCountReducer.java

cd C:\hadoop-3.3.6\wordcount_java

2) Compile the Java Code

javac -classpath C:\hadoop-3.3.6\share\hadoop\common*;C:\hadoop-3.3.6\share\hadoop\mapreduce* -d . WordCountMapper.java WordCountReducer.java WordCountDriver.java

3) Create JAR file

jar -cvf wordcount.jar WordCountDriver.class WordCountMapper.class WordCountReducer.class

#Préparer HDFS

1)make dir in hdfs

hadoop fs -mkdir -p /user_java/input

2) Ajouter un fichier d'entrée

hadoop fs -put input.txt /user_java/input/input.txt

3)**Vérify Files in HDFS** :

hadoop fs -ls /user_java/input

4)Exécute MapReduce program

hadoop jar wordcount.jar WordCountDriver /user_java/input /user_java/output

1)Show Results
hadoop fs -cat /user_java/output/part-r-00000

2)Download Results

hadoop fs -get /user_java/output/part-r-00000 output.txt

manage Erros
hadoop fs -rm -r /user_java/output

hadoop jar wordcount.jar WordCountDriver /user_java/input /user_java/output