

# - INF 729 -

## Rapport sur l'installation d'un cluster Hadoop

### Table des matières

<b>I. Introduction .....</b>	<b>2</b>
<b>II. Environnement technique.....</b>	<b>3</b>
<b>III. Brique HDFS : Système distribué de fichiers .....</b>	<b>4</b>
3.1 Explications autour de HDFS .....	4
3.2 Configuration de HDFS et application d'un programme Wordcount .....	4
<b>IV. Brique MapReduce : Traitement distribué.....</b>	<b>7</b>
4.1 Explication autour de MapReduce .....	7
4.2 Configuration de MapReduce.....	7
<b>V. Zookeeper: Coordination .....</b>	<b>7</b>
5.1 Explication autour de Zookeeper .....	7
5.2 Configuration de Zookeeper .....	8
<b>VI. Hbase : Base de données distribuée .....</b>	<b>9</b>
6.1 Explication autour de Hbase .....	9
6.2 Configuration de Hbase et lancement d'un programme.....	9
<b>VII. Hive : Entrepôt de données .....</b>	<b>10</b>
7.1 Explication autour de Hive .....	10
7.2 Configuration et test de Hive .....	10
<b>VIII. MapReduce - PySpark : Traitements distribués .....</b>	<b>12</b>
8.1 Explication autour de Spark .....	12
8.2 Configuration de Spark pour le traitement distribué.....	12
<b>IX. Conclusion.....</b>	<b>15</b>
<b>X. Annexes .....</b>	<b>16</b>
10.1. Installation d'Hadoop .....	16
10.2. Installation de Zookeeper .....	19
10.3. Installation de HBASE .....	20
10.4. Installation de HIVE.....	21
10.5. Installation de SPark.....	22

## I. Introduction

La principale motivation de l'introduction de l'écosystème Hadoop concerne la gestion des grands volumes de données. En effet, Hadoop est une plateforme open-source capable de gérer d'énormes volumes de données structurées et non structurées et ce dans le cadre d'un système distribué. L'écosystème Hadoop répond à un besoin de Google souhaitant indexer les informations texte collectait afin de présenter des résultats plus pertinents aux utilisateur de son moteur de recherche. Hadoop excelle dans le domaine de l'exécution de tâches nécessitant une grande puissance de calculs et portant sur des données volumineuses.

D'un point de vue de l'architecture, l'écosystème Hadoop a été conçu pour fonctionner sur plusieurs machines (serveurs) de manière simultanée. Chacune des machines offrent donc de la puissance de calcul et du stockage supplémentaire. Concernant le système de gestion des fichiers, les données sont éclatées sur les différentes machines et Hadoop gère un système de réplication de façon à garantir l'accès aux données à n'importe quel moment même lorsqu'une des machines tombe en panne. Concernant la distribution des calculs, la charge des traitements est répartie entre les différentes machines grâce à MapReduce.

De manière plus concrète, l'écosystème Hadoop est composé de trois composants principaux : HDFS, MapReduce et YARN

- HDFS stocke les données (la gestion des gros volumes de données)
- MapReduce fait la magie (la gestion des calculs)
- YARN planifie tout (la gestion des ressources)

Les avantages de Hadoop sont les suivants :

- Traitement des gros volumes de données quel que soit le format
- Mise à l'échelle facile des ressources pour le stockage et le calcul
- Tolérance face aux pannes pour préserver l'intégrité des données
- Simplification de la manipulation des grands volumes de données

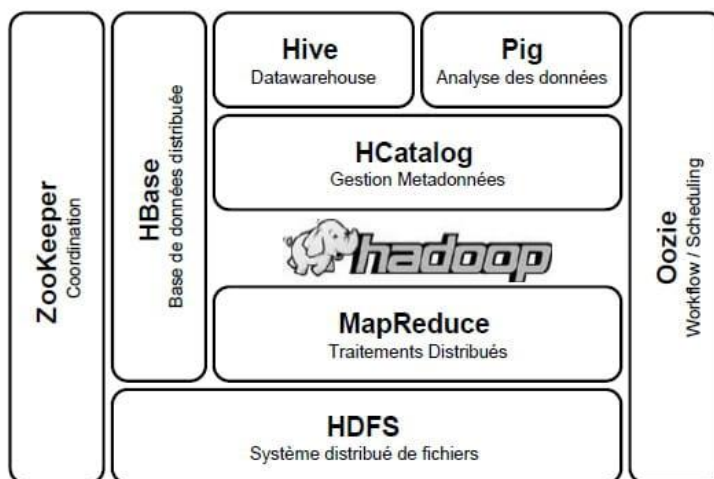


Figure 1: Architecture de l'écosystème Hadoop

Source : <https://www.journaldunet.com/web-tech/developpeur/1125768-panorama-des-solutions-de-big-data/1125771-hadoop-et-son-ecosysteme>

## II. Environnement technique

Pour l'installation et la configuration des clusters Hadoop, nous avons utilisé les machines des salles de travaux pratiques suivantes :

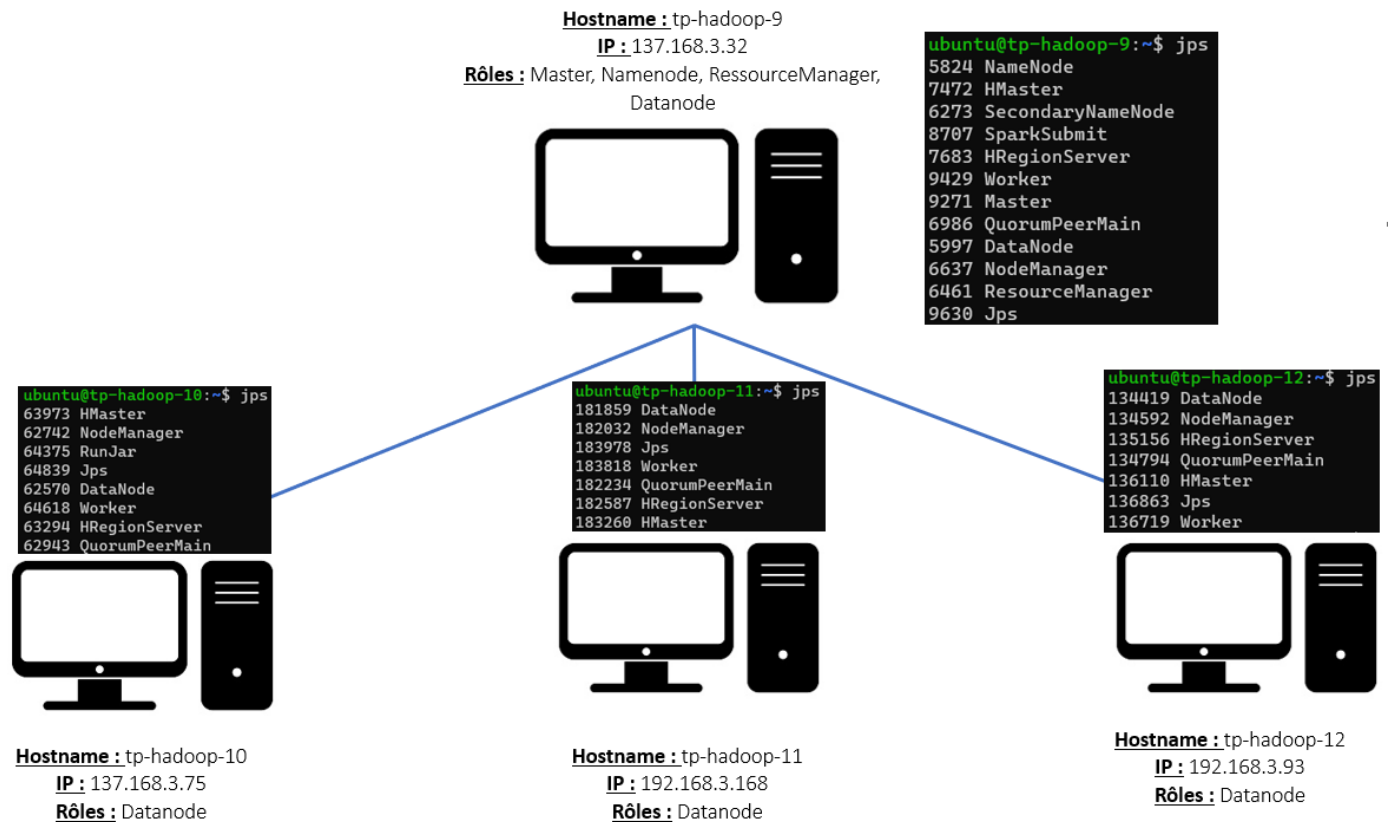
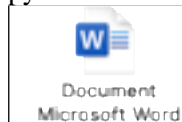


Figure 2: architecture du cluster installé

Pour faciliter le déploiement et le lancement des différentes applications liées à Hadoop, nous avons rédigé un script python « LAUNCH.py » qui centralise l'ensemble des commandes à exécuter sur chaque machine :



### III. Brique HDFS : Système distribué de fichiers

#### 3.1 Explications autour de HDFS

Comme nous l'avons évoqué tout au long de ce rapport, le fait de regrouper des ordinateurs sous forme de cluster permet de répondre aux exigences actuelles de volumes importants de données, d'augmenter les performances en termes de puissance de calcul, de gérer efficacement les montées en charge, d'équilibrer le réseau et d'augmenter le niveau de disponibilité et donc de résilience du système. Le système HDFS est la clé de voute de la tolérance aux pannes du cluster. HDFS (Hadoop Distributed File System) est le système de fichiers distribué et l'élément central de Hadoop permettant de stocker et répliquer des données sur plusieurs serveurs. Il permet de conserver un grand nombre de données sur le long terme ainsi que de les partager entre plusieurs programmes informatiques. HDFS offre à l'utilisateur une vue abstraite sur ses données et permet de les localiser à partir d'un chemin d'accès. Les données sont sérialisées et ce sont ces données qui persistent sur le disque dur en une suites de blocs.

HDFS utilise un NameNode et un DataNode. Le DataNode est un serveur standard sur lequel les données sont stockées. Le NameNode contient des métadonnées (informations sur les données stockées dans les différents nœuds). L'application interagit uniquement avec le NameNode, et celui-ci communique avec les nœuds de données.

La NameNode connaît la répartition exacte des blocs de données d'un fichier en entrée sur l'ensemble des nœuds du cluster. A la différence d'un système de fichier classique, HDFS ne fournit pas de mécanisme de cache (table de référence du système de fichiers, propre à chaque nœud) pour les blocs de fichier. En effet, les blocs de fichier sont beaucoup trop volumineux (64 Mo par défaut) pour être chargés en mémoire. Ainsi à la place, c'est le nœud de référence (NameNode) qui joue le rôle de table de référence partagée. Par conséquent, à chaque requête, les blocs de données sont relus sur le disque dur, ce qui n'est pas vraiment un problème puisque la majorité des applications du MapReduce ne font que des lectures séquentielles. HDFS gère le système de fichier de données de l'ensemble du cluster sous forme de blocs de taille fixe dont le stockage est distribué à travers tous les nœuds du cluster et ce de façon redondante. Pour assurer une haute disponibilité, le système HDFS possède un nœud de référence secondaire qui est prêt à prendre le relai si le nœud de référence tombe en panne. Le nœud de référence second (Secondary Namenode) est une copie du nœud de référence. Ainsi perdre le nœud de référence secondaire signifie perdre le système HDFS et donc perdre le cluster tout entier. Le nœud de référence joue un rôle de « backup » en maintenant une copie de sauvegarde de l'ensemble des données du cluster entier.

#### 3.2 Configuration de HDFS et application d'un programme Wordcount

L'installation de la brique HDFS de l'écosystème Hadoop n'a pas été un long fleuve tranquille. Dans un premier temps nous avons décidé de suivre individuellement les étapes d'installation d'Hadoop sur chaque machine virtuelle, dans le dossier /user/local, tout en créant un « superutilisateur » hadoopuser. Toutefois, les conflits d'installation rencontrés ont porté le groupe vers un choix de centralisation de l'installation, et ce pour la mise en place future de toutes les applications : une seule machine est alors utilisée, puis les dossiers d'applications et fichiers de configuration annexes (ex : .bashrc) sont transmis par commande *scp* aux *workers*. Cela permet d'uniformiser l'édition des fichiers de configuration de l'ensemble des briques logicielle d'hadoop. De cette façon nous évitons les conflits de configuration.

Concernant le choix de l'architecture distribuée nous avons opter pour le choix suivant :

- Une machine a été dédiée au Ressourcemanager et Nodemanager.
- Les trois autres machines sont les workers (DataNode)

Le lancement d'hadoop et de yarn s'effectuent de manière nominale.

```
ubuntu@tp-hadoop-9:~/hadoop$ jps
2976 Jps
1106 DataNode
2486 ResourceManager
1400 SecondaryNameNode
952 NameNode
2648 NodeManager
```

Figure 4: applications java sur le master

```
848 DataNode
1169 Jps
1077 NodeManager
```

Figure 3: applications java sur un worker

Après avoir téléchargé l'archive d'installation d'Hadoop, nous avons édité plusieurs fichiers de configuration propre à HDFS. En effet, tout d'abord, nous avons configuré l'environnement des *daemons* d'Hadoop. Pour ce faire, nous tout d'abord spécifier la variable d'environnement de `JAVA_HOME` sur chaque nœud distant au sein du fichier *hadoop-env.sh*. Ce fichier contient des paramètres de variables d'environnement utilisés par Hadoop. Ensuite nous avons, configuré les paramètres importants de chacun des *daemons* (NameNode) de hadoop en modifiant le fichier *hdfs-site.xml*.

Au sein de ce dernier, nous avons par exemple spécifier le chemin d'accès au dossier du *DataNode* et *NameNode* ainsi que le nombre de réplication de données que nous avons fixé à 2. Par ailleurs dans le prolongement du paramétrage des démons de notre cluster, nous avons spécifié l'URI du *Namenode* accompagné du numéro du port utilise (9000). Enfin, nous avons listé les *hostnames* de chaque *DataNode* dans le fichier *etc/hadoop/workers*.

On note toutefois que l'ensemble des fichiers xml que nous avons édité respectent la structure suivante :

```
<configuration>
  <property>
    <name>...</name>
    <value>...</value>
  </property>
</configuration>
```

A ce stade notre cluster Hadoop n'est pas fonctionnel car il est nécessaire de configurer YARN. YARN est l'abréviation de « Yet Another Resource Negotiator » (plus simplement, un négociateur de ressources). Cet élément assure la gestion et la planification des ressources (clusters) Hadoop et décide de ce qui doit se passer dans chaque nœud de données. Le nœud maître central qui gère toutes les demandes de traitement est le « *Resource Manager* ».

Le Resource Manager interagit avec les différents Node Managers : chaque *DataNode* esclave possède son propre *NodeManager* pour l'exécution des tâches. En clair YARN permet de faire coexister dans la même installation Hadoop plusieurs modèles de calcul distribué. De manière plus concrète, YARN crée pour chaque job du modèle de calcul un container auquel il alloue des ressources et suit l'exécution de ces différentes ressources.

Concernant la configuration de YARN, tout d'abord, nous avons définis un ensemble de variables d'environnement propre à Hadoop tel que « *HADOOP\_HOME* » « *HADOOP\_CONF* », « *HADOOP\_HDFS\_HOME* »... , depuis la machine maître. Enfin nous avons édité le fichier *yarn-site.xml*. Ce fichier contient plusieurs paramètres nécessaires à la gestion des ressources des clusters, dans lequel nous avons spécifié le « *hostname* » de la machine maître. Enfin nous avons testé le bon fonctionnement de notre cluster en affichant le rapport généré par la commande *bin/hadoop dfsadmin -report*.

Pour finir, afin de nous assurer du bon fonctionnement global de notre cluster le groupe a décidé de créer une application *Wordcounter* en JAVA que nous avons récupéré sur le tutoriel d'installation d'un cluster sur le site APACHE HADOOP (<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html>).

Le Wordcounter utilisé est une application java téléchargée sur le site d'Apache puis compilée, à l'aide des commandes suivantes :

#### Création d'un fichier app.java

```
# copier/coller du code dans <app>.java
compilation (dans le dossier hadoop/app) :
    ../bin/hadoop com.sun.tools.javac.Main <app>.java
    jar cf <app>.jar WordCount*.class
copie dans le dossier input de hdfs : bin/hadoop fs -put <file> input
lancer l'app.jar : bin/hadoop jar <app>.jar app input output
```

Choix des données d'entrée :

- CommonCrawl répliqué 10 fois : fichier texte de 3.6 Go:
- <https://commoncrawl.s3.amazonaws.com/crawl-data/CC-MAIN-2017-13/segments/1490218189495.77/wet/CC-MAIN-20170322212949-00140-ip-10-233-31-227.ec2.internal.warc.wet.gz>
- Page de téléchargement de fichier wikipedia : <https://dumps.wikimedia.org/enwiki/20211101/>

Le temps de MapReduce pour le fichier de 3,6 Go s'établit à 15 minutes environ.

```
2021-11-18 19:50:00,756 INFO mapred.LocalJobRunner: reduce task executor complete.
2021-11-18 19:50:00,954 INFO mapreduce.Job: map 100% reduce 100%
2021-11-18 19:50:00,954 INFO mapreduce.Job: Job job_local602551035_0001 completed successfully
2021-11-18 19:50:00,977 INFO mapreduce.Job: Counters: 36

File System Counters
  FILE: Number of bytes read=26325179479
  FILE: Number of bytes written=44058086715
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=54290340400
  HDFS: Number of bytes written=102873163
  HDFS: Number of read operations=899
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=30
  HDFS: Number of bytes read erasure-coded=0

Map-Reduce Framework
  Map input records=87531471
  Map output records=432142218
  Map output bytes=5310206478
  Map output materialized bytes=1250308929
  Input split bytes=3051
  Combine input records=506886282
  Combine output records=129018683
  Reduce input groups=4927784
  Reduce shuffle bytes=1250308929
  Reduce input records=54274619
  Reduce output records=4927784
  Spilled Records=183293302
  Shuffled Maps =27
  Failed Shuffles=0
  Merged Map outputs=27
  GC time elapsed (ms)=5894
  Total committed heap usage (bytes)=19095093248

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=3589178648
File Output Format Counters
  Bytes Written=102873163
```

Figure 5: script de fin de programme wordcount

En conclusion, comme évoqué précédemment l'installation de notre cluster Hadoop n'a pas été une tâche facile à réaliser. En effet, nous avons dû réinitialiser notre machine virtuelle à deux reprises pour faire face à des conflits de configuration entre chacune des machines esclaves (worker). A cela s'ajoute le fait que l'ensemble des membres du groupe découvrait pour la 1<sup>ère</sup> fois la manipulation des commandes bash. Cependant nous avons finalement réussi à mettre en œuvre un cluster Hadoop multi node fonctionnel.

## IV. Brique MapReduce : Traitement distribué

### 4.1 Explication autour de MapReduce

MapReduce est un modèle de programmation qui a d'abord été utilisé par Google pour indexer ses opérations de recherche. Suivant cette logique, cet élément exécute des algorithmes pour décomposer des données en datasets plus petits. MapReduce s'appuie sur deux fonctions : **Map()** et **Reduce()**, qui analysent les données rapidement et efficacement.

La fonction Map regroupe, filtre et trie plusieurs datasets en parallèle et génère des tuples (paires key/value). La fonction Reduce agrège ensuite les données de ces tuples pour produire le résultat souhaité.

Le schéma d'exécution des calculs en parallèle est basé :

- sur deux étapes : le mappage et la réduction,
  - sur les concepts de la programmation fonctionnelle (1995).
1. **Mapping** : les nœuds ouvriers appliquent la fonction de mappage aux données locales ; la sortie est écrite dans le fichier temporaire. Le nœud maître s'assure qu'une seule copie est traitée.
  2. **Shuffling** : les nœuds ouvriers redistribuent les données en fonction des clés de sortie (produites par la fonction map), de sorte que toutes les données appartenant à une clé se trouvent sur le même nœud esclave.
  3. **Reducing** : les nœuds esclave traitent chaque groupe de données de sortie, par clé, en parallèle.

### 4.2 Configuration de MapReduce

Concernant la configuration de la brique logicielle MapReduce, nous avons modifié le fichier *etc/hadoop/mapred-site.xml*. Ce fichier contient des informations de configuration qui remplacent les valeurs par défaut des paramètres MapReduce. Ainsi, dans ce fichier nous avons spécifier le framework d'exécution utilisé par MapReduce qui est YARN ainsi que la variable d'environnement HADOOP\_MAPRED\_HOME

## V. Zookeeper: Coordination

### 5.1 Explication autour de Zookeeper

ZooKeeper permet aux processus distribués de se coordonner entre eux via un espace de noms de journal de données hiérarchique partagé, appelé znodes. Chaque znode est identifié par un chemin, où les éléments du chemin sont séparés par une barre oblique ("/"). En plus de l'élément racine, chaque znode a un élément de niveau supérieur ; De plus, un znode ne peut pas être supprimé s'il a des éléments enfants.

Il est très similaire à un système de fichiers normal, mais rendu très fiable par les services redondants proposés par ZooKeeper. Chaque service est répliqué sur un ensemble de machines, et tous les ensembles conservent une image de l'arborescence des données et des journaux de transactions dans leur mémoire.

Les clients se connectent à un seul serveur ZooKeeper et maintiennent une connexion TCP via laquelle ils envoient des demandes et reçoivent des réponses.

Cette architecture permet à ZooKeeper de fournir un débit et une disponibilité élevés avec une faible latence, bien que la taille de la base de données que ZooKeeper peut gérer soit limitée par la mémoire.



## 5.2 Configuration de Zookeeper

```
ubuntu@tp-hadoop-11:~/zookeeper/conf$ cat zoo.cfg
tickTime=2000
initLimit=5
syncLimit=2
dataDir=/home/ubuntu/zookeeper/data
dataLogDir=/home/ubuntu/zookeeper/logs
clientPort=2181
autopurge.snapRetainCount=10
autopurge.purgeInterval=24
server.1=tp-hadoop-9:2888:3888
server.2=tp-hadoop-10:2888:3888
server.3=tp-hadoop-11:2888:3888
server.4=tp-hadoop-12:2888:3888
ubuntu@tp-hadoop-11:~/zookeeper/conf$
```

Figure 6: configuration de Zookeeper

Après téléchargement des différents éléments de Zookeeper ainsi que de leur extraction ; les fichiers de configuration se trouveront dans le répertoire `~/zookeeper/conf`. Ce répertoire contient un exemple de fichier de configuration fourni avec la distribution ZooKeeper. Cet exemple de fichier, nommé `zoo_sample.cfg`, contient les définitions de paramètres de configuration les plus courantes ainsi que des exemples de valeurs pour ces paramètres.

Ces paramètres utilisés sont les suivants :

- *tickTime* = 2000  
Définit le temps d'un tick en millisecondes. Un tick est une unité de temps utilisée par ZooKeeper pour mesurer le temps écoulé entre deux battements de cœur. Les délais d'expiration de session minimum sont le double du *tickTime*.
- *initLimit* = 10  
C'est la durée, en ticks, pour permettre aux followers de se connecter et de se synchroniser avec un leader. Il est possible d'augmenter cette valeur si nécessaire, si la quantité de données gérées par ZooKeeper est importante.
- *syncLimit* = 5  
Il s'agit de la durée maximale pendant laquelle les nœuds peuvent être désynchronisés par rapport au leader.
- *dataDir* = */home/ubuntu/zookeeper/data*  
Spécifie le répertoire utilisé pour stocker les snapshots de la base de données en mémoire et le journal des transactions pour les mises à jour. Il est possible de choisir de spécifier un répertoire distinct pour les journaux de transactions.
- *dataLogDir* = */home/ubuntu/zookeeper/logs*  
Cette option ordonne à la machine d'écrire dans la log transaction *dataLogDir* plutôt que dans le *dataDir*. Cela permet d'utiliser un périphérique de journalisation dédié et d'éviter la concurrence entre les logging et les snapshots.
- *clientPort* = 2181  
Port utilisé pour suivre les connexions client.
- *autopurge.snapRetainCount* = 10  
Lorsqu'elle est activée, la fonction de purge automatique de ZooKeeper conserve les snapshots les plus récents (ici les 10 plus récents) ainsi que leur log transaction correspondants dans les *dataDir* et *dataLogDir*.



- `autopurge.purgeInterval = 24`  
L'intervalle de temps en heures pour lequel la tâche de purge doit être déclenchée. Il faut définir un nombre entier positif (1 et plus) pour activer la purge automatique. La valeur par défaut est 0.
- `server.< nb ≥ < hostname 1 >:2888:3888`  
Les nœuds ZooKeeper utilisent une paire de ports, :2888 et :3888, pour que les nœuds suiveurs se connectent au nœud leader et également pour l'élection du leader.

## VI. Hbase : Base de données distribuée

### 6.1 Explication autour de Hbase

HBase est une base de données non relationnelles en colonnes qui s'installe en surcouche du HDFS. L'un des défis du HDFS est qu'il est limité au traitement par lots. Autrement dit, pour les requêtes interactives simples, les données doivent quand même être traitées par lots, ce qui entraîne une latence élevée.

HBase contourne cette situation en supportant les requêtes portant sur une seule ligne, y compris dans les tables les plus volumineuses, ce qui réduit considérablement les temps de latence. Ce processus fait appel à des tables de hachage en interne ; il est modélisé sur Google BigTable qui permet d'accéder au système de fichiers Google (GFS).

HBase est évolutif, réagit efficacement aux incidents (par exemple, si un nœud tombe en panne) et il peut traiter les données non structurées et les données semi-structurées. C'est donc la solution idéale pour interroger de gros gisements de données à des fins d'analyse.

### 2.2 Configuration de Hbase et lancement d'un programme

HBase est paramétré pour prendre en charge directement Zookeeper, ce qui limite les gestions des couches supplémentaires rajoutées à Hadoop. En suivant le récapitulatif placé en annexe, nous n'avons pas rencontré de problèmes sur la phase d'installation.

Cette couche supplémentaire pourrait permettre d'améliorer le calcul si les fichiers téléchargés sont implémentés en tant que tables dans HBase via l'interface de HIVE.

L'installation d'Hbase n'a pas posé de soucis. Son lancement est affiché ci-dessous en guise d'exemple :

```
ubuntu@tp-hadoop-9:~/hbase/bin$ ./start-hbase.sh
tp-hadoop-11: running zookeeper, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-zookeeper-tp-hadoop-11.out
tp-hadoop-10: running zookeeper, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-zookeeper-tp-hadoop-10.out
tp-hadoop-12: running zookeeper, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-zookeeper-tp-hadoop-12.out
tp-hadoop-9: running zookeeper, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-zookeeper-tp-hadoop-9.out
running master, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-master-tp-hadoop-9.out
tp-hadoop-10: running regionserver, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-regionserver-tp-hadoop-10.out
tp-hadoop-9: running regionserver, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-regionserver-tp-hadoop-9.out
tp-hadoop-12: running regionserver, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-regionserver-tp-hadoop-12.out
tp-hadoop-11: running regionserver, logging to /home/ubuntu/hbase/logs/hbase-ubuntu-regionserver-tp-hadoop-11.out
ubuntu@tp-hadoop-9:~/hbase/bin$ jps
1106 DataNode
2486 ResourceManager
1400 SecondaryNameNode
952 NameNode
2648 NodeManager
4555 Jps
4124 HQuorumPeer
4445 HRegionServer
4239 HMaster
```

Figure 7: Lancement d'HBase sur le master

## VII. Hive : Entrepôt de données

### 7.1 Explication autour de Hive

Hive est un système de data warehousing qui permet d'interroger les gros datasets présents dans le HDFS. Avant que Hive apparaisse, les développeurs étaient confrontés au défi de créer des tâches MapReduce complexes pour interroger les données Hadoop. Hive utilise la langage HQL (Hive Query Language), dont la syntaxe est semblable à celle de SQL. La plupart des développeurs ayant l'habitude des environnements et du langage SQL, sont rapidement à l'aise avec Hive.

L'un des avantages de Hive est que l'interface entre l'application et le HDFS est assurée par un pilote JDBC/ODBC. Ce pilote affiche le système de fichiers Hadoop sous forme de tables et convertit le HQL en job MapReduce (et vice-versa). Les développeurs et les administrateurs de bases de données bénéficient ainsi du meilleur des deux mondes : les avantages du traitement par lots des gros datasets et la possibilité de formuler des requêtes simples dans un environnement familier. Développé à l'origine par l'équipe Facebook, Hive est aujourd'hui une technologie open source.

Motivations de l'introduction de Hive :

- Développé chez Facebook en 2007
- Code MapReduce difficile à écrire
- Abstraction SQL pour MapReduce Permet d'écrire des requêtes de type SQL pour les tâches MapReduce.
- Hive n'est pas une base de données complète
- Il peut être interfacé avec HBase

Le langage de requête pour s'interfacer avec MapReduce

- Types simples : *int*, *bool*, *float*
- Types de collections : *struct*, *map*, *array*

### 7.2 Configuration et test de Hive

L'installation de Hive a posé des problèmes de collision de fichiers java, en doublons avec les fichiers java déjà présents dans Hadoop. Les résolutions que nous avons mis en œuvre pour effacer l'apparition des messages d'erreurs java au lancement de Hive sont les suivantes :

- SLF4J: Found binding in [jar:file:/home/ubuntu/hive/lib/slf4j-log4j12-1.7.30.jar... : supprimer le fichier
- com.ctc.wstx.exc.WstxParsingException: Illegal character entity: expansion character (code 0x8 at [row,col,system-id]: [3215,96,"file:/home/ubuntu/hive/conf/hive-site.xml"]) : supprimer le caractère dans le fichier (rechercher « exclusive locks for... » et supprimer les caractères spéciaux juste après.
- Exception in thread "main" java.lang.IllegalArgumentException: java.net.URISyntaxException: Relative path in absolute URI: \${system:java.io.tmpdir%7D/%7Bsystem:user.name%7D : insérer juste après <configuration> de hive-site.xml les lignes suivantes :

```
<property>
  <name>system:java.io.tmpdir</name>
  <value>/tmp/hive/java</value>
</property>
<property>
  <name>system:user.name</name>
  <value>${user.name}</value>
</property>
```

- hive > FAILED: HiveException java.lang.RuntimeException: Unable to instantiate org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient : modifier le fichier sql comme suit :

```
nano ~/hive/scripts/metastore/upgrade/derby/hive-schema-3.1.0.derby.sql
commenter les deux premières lignes de définition de
#CREATE FUNCTION "APP"."NUCLEUS_ASCII"
#CREATE FUNCTION "APP"."NUCLEUS_MATCHES"
Relancer $HIVE_HOME/bin/schematool -dbType derby -initSchema
```

Application : Hive est utilisé en tant que wordcount par la création d'une nouvelle table comme suit :

```
CREATE TABLE FILES (line STRING);
LOAD DATA INPATH 'input/<fichier d'input>' OVERWRITE INTO TABLE FILES;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, ' ')) AS word FROM FILES) w
GROUP BY word
ORDER BY word;
```

Le résultat pour le fichier de 3.3 Go est de 271 secondes.

```
Query ID = ubuntu_20211109103415_aebab42f-a04f-4e23-9946-f69c8a351560
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 2
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2021-11-09 10:34:21,032 Stage-1 map = 0%, reduce = 0%
2021-11-09 10:35:21,116 Stage-1 map = 0%, reduce = 0%
2021-11-09 10:35:40,060 Stage-1 map = 17%, reduce = 0%
2021-11-09 10:36:40,087 Stage-1 map = 17%, reduce = 0%
2021-11-09 10:36:52,403 Stage-1 map = 46%, reduce = 0%
2021-11-09 10:36:54,460 Stage-1 map = 100%, reduce = 0%
2021-11-09 10:37:06,214 Stage-1 map = 50%, reduce = 0%
2021-11-09 10:38:07,307 Stage-1 map = 50%, reduce = 0%
2021-11-09 10:38:12,605 Stage-1 map = 83%, reduce = 0%
2021-11-09 10:38:16,616 Stage-1 map = 100%, reduce = 0%
2021-11-09 10:38:23,631 Stage-1 map = 100%, reduce = 50%
2021-11-09 10:38:29,643 Stage-1 map = 100%, reduce = 100%
Ended Job = job_local1102551217_0001
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2021-11-09 10:38:31,246 Stage-2 map = 0%, reduce = 0%
2021-11-09 10:38:42,497 Stage-2 map = 33%, reduce = 0%
2021-11-09 10:38:48,504 Stage-2 map = 87%, reduce = 0%
2021-11-09 10:38:54,512 Stage-2 map = 100%, reduce = 0%
2021-11-09 10:39:07,650 Stage-2 map = 100%, reduce = 100%
Ended Job = job_local416655098_0002
Moving data to directory hdfs://tp-hadoop-9:9000/user/hive/warehouse/word_counts
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 1460943232 HDFS Write: 0 SUCCESS
Stage-Stage-2: HDFS Read: 797582528 HDFS Write: 102866871 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 292.494 seconds
```

Figure 8: résultat de calcul wordcount

Le résultat peut être lu à partir de la commande suivante :  
bin/hadoop fs -cat /user/hive/warehouse/word\_counts/000000\_0

## VIII. MapReduce - PySpark : Traitements distribués

### 8.1 Explication autour de Spark

Spark est une alternative plus efficace à MapReduce. En effet, MapReduce est bâti sur un modèle de flux acyclique direct qui n'est pas adapté à certaines applications et en particulier les applications qui réutilisent les données à travers de multiples opérations comme cela est le cas pour la plupart des algorithmes de machines learning. Spark a pour objectif de gommer les limites de MapReduce.

Par ailleurs, Spark n'est pas un paradigme de programmation comme peut l'être MapReduce. En effet, ce dernier est un moteur de calcul distribué qui supporte les applications itératives tout en conservant la scalabilité et la tolérance aux pannes d'un cluster Hadoop. Autrement dit, Spark est une couche logicielle installée sur un cluster lui permettant d'effectuer des traitements de données en mémoire tout en conservant les attributs de scalabilité. Aussi ce dernier est potentiellement 100 fois plus rapide que Hadoop MapReduce car il utilise la RAM et n'est pas lié au paradigme à deux étapes d'Hadoop.

En effet, les calculs effectués par Spark sont exécutés dans la mémoire RAM de chaque nœud de données. Pour atteindre ce but, Spark utilise une abstraction que l'on appelle RDD. Un RDD est une collection d'objet immuable (lecture seule) partitionnée et distribuée à travers les nœuds d'un cluster. Les RDD sont conservés en mémoire dans les nœuds du cluster et réutilisés dans chaque opération parallélisée. Les RDD sont résilients aux pannes car ils peuvent être reconstruits automatiquement grâce à la fonctionnalité Spark Lineage. Spark Lineage construit un graphe de dépendances entre le RDD existant et le nouveau RDD. Cela signifie que toutes les dépendances entre les RDD sont enregistrées dans un graphe plutôt que d'enregistrer les données d'origines ainsi que leurs modifications.

En fait, Apache Spark fonctionne bien pour les petits ensembles de données qui peuvent tous tenir dans la RAM d'un serveur. Hadoop MapReduce est plus rentable en terme coût pour le traitement d'ensembles de données massifs. De plus, Spark est adapté au traitement de données en temps réel. Dorénavant, Apache Spark est désormais plus populaire que Hadoop MapReduce.

### 8.2 Configuration de Spark pour le traitement distribué

Après avoir téléchargé et dézippé l'archive d'installation de Spark sur la machine maître pour la configuration de Spark nous avons suivi globalement les mêmes étapes que le paramétrage du cluster Hadoop. En effet, nous avons tout d'abord défini les variables d'environnement de Spark (*SPARK\_HOME*) dans le fichier *.bashrc*.

Par la suite nous avons paramétré Spark en éditant le fichier *saprk-env.sh*. Dans ce fichier nous avons défini la variable d'environnement utilisée par Java (*JAVA\_HOME*) et nous avons également spécifier l'adresse IP de la machine maître. Enfin, nous avons terminé la configuration de Spark en créant un fichier slaves dans le dossier *~/spark/conf/slaves* afin d'ajouter le hostname de la machine maître et des machines esclaves (*ex : tp-hadoop-9 # master, tp-hadoop-10 # slave,...*).

Pour démarrer les démons spark nous exécutons la ligne suivante *~/spark/sbin/start-all.sh*. Une fois les démons lancés lorsque nous tapons la commande *jps*, l'attribut Worker apparaît sur chacune des machines.

Pour tester le fonctionnement de Spark nous avons dans un premier temps exécuté un programme Wordcount rédigé en PySpark. Pour l'exécution de ce programme nous avons d'abord créé un dossier *WC\_pyspark* dans le dossier *~/spark* puis dans ce dossier nous avons créé le fichier Wordcount portant le nom suivant : *WC\_pyspark\_file\_arg.py*.

Ce fichier pour être exécuté prend en argument un fichier txt que nous avons préalablement stocké dans le cluster hadoop (*./hadoop/data/file02*) :

```
*Exécution du fichier "WC_pyspark_file_arg.py"*
# depuis la racine du dossier Spark (~/spark)
bin/spark-submit ./WC_pyspark/WC_pyspark_file_arg.py ./hadoop/data/file02
```

WC\_pyspark\_file\_arg.py

```
import sys
from operator import add

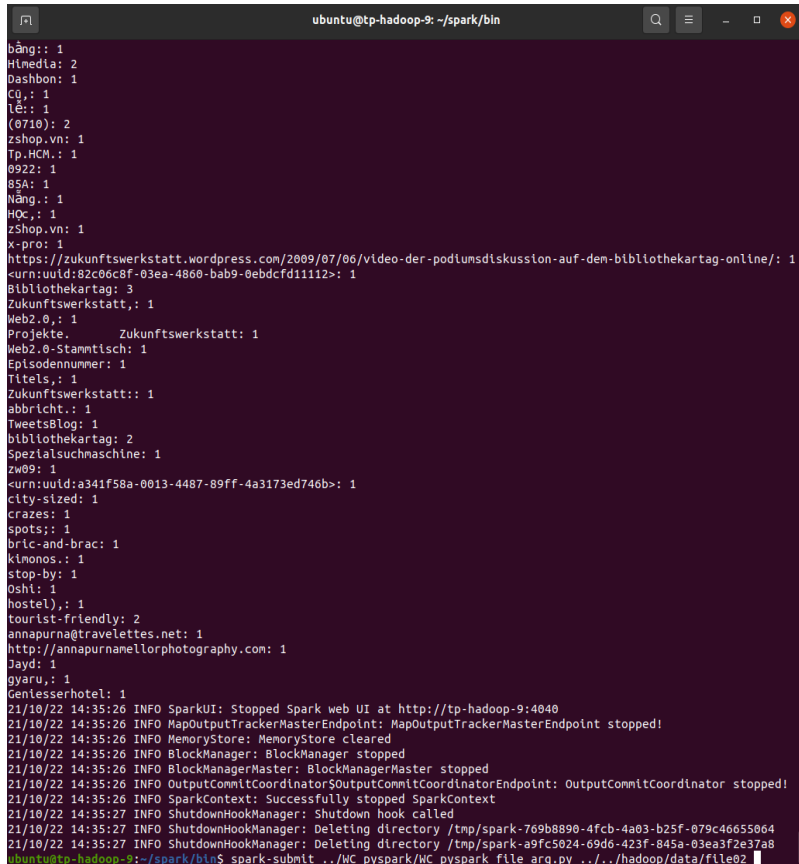
from pyspark.sql import SparkSession

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        sys.exit(-1)

    spark = SparkSession\
        .builder\
        .appName("PythonWordCount")\
        .getOrCreate()

    lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
    counts = lines.flatMap(lambda x: x.split(' ')) \
        .map(lambda x: (x, 1)) \
        .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))

    spark.stop()
```



```
ubuntu@tp-hadoop-9: ~/spark/bin
bâng: 1
Himedia: 2
Dashbon: 1
Cù: 1
lê: 1
(0710): 2
zshop.vn: 1
Tp.HCM: 1
0922: 1
BSA: 1
Nâng: 1
HOC: 1
zshop.vn: 1
X-pro: 1
https://zukunftswerkstatt.wordpress.com/2009/07/06/video-der-podiumsdiskussion-auf-den-bibliothekartag-online/: 1
&urn:uuid:82c06c8f-03ea-4860-bab9-0ebdcfd11112>: 1
Bibliothekartag: 3
Zukunftswerkstatt: 1
Web2.0: 1
Projekte. Zukunftswerkstatt: 1
Web2.0-Stammtisch: 1
Episodennummer: 1
Titels: 1
Zukunftswerkstatt: 1
abbricht.: 1
TweetsBlog: 1
bibliothekartag: 2
Spezialsuchmaschine: 1
zw09: 1
&urn:uuid:a341f58a-0013-4487-89ff-4a3173ed746b>: 1
city-sized: 1
crazes: 1
spots: 1
bric-and-brac: 1
klmonos.: 1
stop-by: 1
Oshl: 1
hostel): 1
tourist-friendly: 2
annapurna@travelettes.net: 1
http://annapurnanellorphotography.com: 1
Jayd: 1
gyaru: 1
Gentleserhotel: 1
21/10/22 14:35:26 INFO SparkUI: Stopped Spark web UI at http://tp-hadoop-9:4040
21/10/22 14:35:26 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/10/22 14:35:26 INFO MemoryStore: MemoryStore cleared
21/10/22 14:35:26 INFO BlockManager: BlockManager stopped
21/10/22 14:35:26 INFO BlockManagerMaster: BlockManagerMaster stopped
21/10/22 14:35:26 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
21/10/22 14:35:26 INFO SparkContext: Successfully stopped SparkContext
21/10/22 14:35:27 INFO ShutdownHookManager: Shutdown hook called
21/10/22 14:35:27 INFO ShutdownHookManager: Deleting directory /tmp/spark-769b8890-4fcb-4a03-b25f-079c46655064
21/10/22 14:35:27 INFO ShutdownHookManager: Deleting directory /tmp/spark-a9fc5024-69d6-423f-845a-03ea3f2e37a8
ubuntu@tp-hadoop-9:~/spark/bin$ spark-submit ../WC_pyspark/WC_pyspark_file_arg.py ../hadoop/data/file02
```

Figure 9: résultat de wordcount par Spark

Jusqu'ici nous avons exécuté un programme dont le fichier se trouve dans le dossier de la machine sur lequel le programme Wordcount pyspark est exécuté. Par conséquent ici, le programme n'utilise pas la puissance des 3 machines esclaves pour exécuter son job.

Or dans notre cas, nous souhaitons exécuter un programme PySpark utilisant comme donnée d'entrée, un fichier stocké dans un nœud du cluster. En procédant de cette manière cela nous permettra de nous assurer que Spark tire profit de la puissance des 3 machines pour exécuter son job en parallèle sur ces dernières.

Ce fichier est le même fichier utilisé par le programme MapReduce à savoir, le fichier provenant du site CommonCrawl qui a été répliqué 10 fois. Ce fichier a une taille de 3.6 Go. Le code de notre programme pySpark se trouve ci-dessous.

## Rapport INF 729 – Installation d'un cluster Hadoop

Pour fournir en entrée le fichier stocké dans notre cluster hadoop à notre programme Wordcount nous avons ajouté la ligne suivante :

```
sc.testFile("hdfs://tp-hadoop-9:9000/user/ubuntu/input/file02")
```

```
GNU nano 4.8 wc.py
import sys

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    # create Spark context with necessary configuration
    sc = SparkContext("local", "PySpark Word Count Example")

    # read data from text file and split each line into words
    words = sc.textFile("hdfs://tp-hadoop-9:9000/user/ubuntu/input/file02").flatMap(lambda line: line.split(" "))

    # count the occurrence of each word
    wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)

    # save the counts to output
    wordCounts.saveAsTextFile("hdfs://tp-hadoop-9:9000/user/ubuntu/sparkoutput")
```

Figure 10: wc.py

```
21/11/19 15:44:38 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
21/11/19 15:44:38 INFO SparkContext: Successfully stopped SparkContext
21/11/19 15:44:38 INFO ShutdownHookManager: Shutdown hook called
21/11/19 15:44:38 INFO ShutdownHookManager: Deleting directory /tmp/spark-e89c3f1e-7e8f-42bd-b1cb-32f0487d95a7/pyspark-4b4f1acb-ced7-4525-b961-f2e7148a4222
21/11/19 15:44:38 INFO ShutdownHookManager: Deleting directory /tmp/spark-e89c3f1e-7e8f-42bd-b1cb-32f0487d95a7
21/11/19 15:44:38 INFO ShutdownHookManager: Deleting directory /tmp/spark-79638108-be6e-468a-83f1-7f0c788f7873
ubuntu@tp-hadoop-12:~/spark/WC_pyspark$ cd
ubuntu@tp-hadoop-12:~$ cd hadoop/
ubuntu@tp-hadoop-12:~/hadoop$ bin/hadoop fs -ls
Found 4 items
drwxr-xr-x - ubuntu supergroup          0 2021-11-18 20:23 input
drwxr-xr-x - ubuntu supergroup          0 2021-11-18 19:50 output
drwxr-xr-x - ubuntu supergroup          0 2021-11-18 20:18 output2
drwxr-xr-x - ubuntu supergroup          0 2021-11-19 15:44 sparkoutput
ubuntu@tp-hadoop-12:~/hadoop$ bin/hadoop fs -ls sparkoutput
Found 4 items
-rw-r--r--  3 ubuntu supergroup          0 2021-11-19 15:44 sparkoutput/_SUCCESS
-rw-r--r--  3 ubuntu supergroup 42683746 2021-11-19 15:44 sparkoutput/part-00000
-rw-r--r--  3 ubuntu supergroup 43135986 2021-11-19 15:44 sparkoutput/part-00001
-rw-r--r--  3 ubuntu supergroup 42837344 2021-11-19 15:44 sparkoutput/part-00002
ubuntu@tp-hadoop-12:~/hadoop$ cat part-00002
```

Figure 11: Trace des outputs générés lors de l'exécution du job pySpark

```
ubuntu@tp-hadoop-12:~/hadoop$ bin/hadoop fs -head sparkoutput/part-00002
('WARC-Type:', 45301)
('warinfo', 1)
('2017-04-03T06:37:57Z', 1)
('urn:uuid:fa8d394b-1efb-4f8b-b478-68dd5e40d75d', 1)
('Content-Type:', 45317)
('Content-Length:', 45307)
('Apr', 9641)
('2017', 51238)
('06:37:57', 1)
('isPartOf:', 1)
('CC-MAIN-2017-13', 1)
('publisher:', 12)
('http://01free-share.com/yYS', 2)
('sha1:IDY4L24RTBWIR6WLHML7VMRIFZKWI5Z6', 1)
('text/plain', 45307)
('1457', 35)
('01free-share.Com', 1)
('About', 24005)
('Use', 8128)
('our', 34010)
('API', 1191)
('Stats', 1923)
('Contact', 24804)
('01share.com', 1)
('a', 374839)
('redirection', 1)
('anyone', 2069)
('take', 8419)
('shorten', 34)
('it.', 5708)
('type/paste', 1)
('in', 285092)
('on', 146178)
('site', 10504)
('main', 4240)
('page', 9133)
('long', 5033)
('You', 27816)
```

Figure 12: Extrait du résultat généré par le programme Spark



## IX. Conclusion

L'installation et la configuration de l'ensemble des briques logicielles constituant l'écosystème Hadoop n'a pas été un long fleuve tranquille. Cependant grâce à cet enseignement nous sommes en mesure de mettre en place un écosystème Hadoop accompagné de ses principales fonctionnalités majeures comme Zookeeper, Hbase, Hive et Spark.

Par ailleurs nous avons pu nous apercevoir à travers les différents programme Wordcount que nous avons exécuté sur notre cluster de la puissance des système distribués.

Enfin, au-delà de nous familiariser avec le fonctionnement de l'écosystème Hadoop en ayant un aperçu global de l'architecture ainsi que les interactions qu'ils existes entre les différentes briques logicielles, cet enseignement nous a permis de nous familiariser avec les commandes shell et ainsi ne plus avoir peur à éditer des fichiers de configuration en ligne de commande.



## X. Annexes

Accès au Cluster :

**Réseau école** : `ssh <login>@ssh.enst.fr`

**Bridge** : `ssh ubuntu@137.194.211.146`

Connection machine : `ssh tp-hadoop-<No>`

### 10.1. Installation d'Hadoop



Sur chaque machine :

Vérification de mise à jour de l'environnement :

```
sudo apt update
sudo apt upgrade
sudo apt install ssh
sudo apt install pdsh
sudo su
echo "ssh" > /etc/pdsh/rcmd_default
sudo apt install openjdk-8-jdk
```

Vérification du fichier hostname et hosts :

Dans hosts : `<adresseip des machines du cluster> <hostname>`

Dans hostname : `<nom machine>`

Générer clé publique : `ssh-keygen -t rsa -P ""`

Enregistrement clé pub : `cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys`

Test local host : `ssh localhost`

Si mot de passe demandé :

```
ssh-keygen -t rsa -P " " -f ~/.ssh/id_rsa
```

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
chmod 0600 ~/.ssh/authorized_keys
```

Téléchargement et décompression :

```
cd
wget https://mirrors.sonic.net/apache/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz && tar -zxvf hadoop-3.3.1.tar.gz && rm -rf hadoop-3.3.1.tar.gz
mv hadoop-3.3.1/ hadoop
cd hadoop
mkdir -p data/dataNode data/nameNode
```

Sur la machine maitre :

```
cd hadoop
nano etc/hadoop/hadoop-env.sh → export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
nano etc/hadoop/core-site.xml
```

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://<hostname machine maître>:9000</value>

</property>

</configuration>

```
nano etc/hadoop/hdfs-site.xml
<configuration>
<property>
<name>dfs.namenode.name.dir</name>
<value>file://<chemindata/nameNode></value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file://<chemindata/dataNode></value>
</property>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
</configuration>
    nano etc/hadoop/mapred-site.xml
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
<property>
<name>mapreduce.application.classpath</name>
<value>${HADOOP_MAPRED_HOME}/share/hadoop/mapreduce/*:${HADOOP_MAPRED_HOME}/share/hadoop/ma
preduce/lib/*</value>
</property>
</configuration>
nano etc/hadoop/yarn-site.xml
<property>
<name>yarn.resourcemanager.hostname</name>
<value><hostnamemaster></value>
</property>
</configuration>
```

nano etc/hadoop/workers : lister les hostnames des datanodes

envoi des conf aux workers : scp ~/hadoop/etc/hadoop/\* <worker>:~/hadoop/etc/hadoop/

Paramétrage de Yarn sur la machine maitre :

```
export HADOOP_HOME="/~/hadoop"
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
```

Dans etc/hadoop/yarn-site.xml :

```
<property>
<name>yarn.resourcemanager.hostname</name>
<value>tp-hadoop-9</value>
</property>
```

formatage en HDFS : hdfs namenode -format

### Test du cluster :

démarrage hdfs : `sbin/start-dfs.sh`  
démarrage yarn : `sbin/start-yarn.sh`

vérification sur API : `http://137.194.211.146/<maitre>/9870/dfshealth.html`  
vérification par terminal (sur chaque machine) : `jps`  
vérification des logs : `cat logs/*.log`  
vérification par rapport : `bin/hadoop dfsadmin -report`

création d'un dossier de travail sur hdfs : `bin/hadoop fs -mkdir -p /user/<username>`  
création du dossier input : `bin/hadoop fs -mkdir -p input`

Lancement d'un appli Wordcount : `bin/hadoop fs jar WC/wc.jar WordCount input/file01 outputX`

Rappel des variables à mettre dans `.bashrc`  
`export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64/"`  
`export PATH=${JAVA_HOME}/bin:${PATH}`  
`export HIVE_HOME="/home/ubuntu/hive"`  
`export PATH=$PATH:$HIVE_HOME/bin`  
`export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.`  
`export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.`  
`export DERBY_HOME="/home/ubuntu/derby"`  
`export PATH=$PATH:$DERBY_HOME/bin`  
`export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar`

arrêt hdfs : `sbin/stop-dfs.sh`  
arrêt yarn : `sbin/stop-yarn.sh`

Affichage des rapports : `bin/hadoop fs -report`

au besoin :

suppression de la partition hdfs : `rm -rf /tmp/hadoop-<username>/dfs`  
trouver le clusterId : `cat ~/hadoop/data/nameNode/current`



## 10.2. Installation de Zookeeper

Téléchargement de zookeeper : `wget https://mirroir.wptheme.fr/apache/zookeeper/zookeeper-3.6.3/apache-zookeeper-3.6.3-bin.tar.gz && tar -zxvf apache-zookeeper-3.6.3-bin.tar.gz && mv apache-zookeeper-3.6.3-bin zookeeper && rm -rf apache-zookeeper-3.6.3-bin.tar.gz`

Création d'un fichier de config en stand alone : `cd zookeeper && nano conf/zoo.cfg`  
`tickTime=2000`

`initLimit=10`

`syncLimit=5`

`dataDir=/home/ubuntu/zookeeper/data`

`dataLogDir=/home/ubuntu/zookeeper/logs`

`clientPort=2181`

`autopurge.snapRetainCount=10`

`autopurge.purgeInterval=24`

`server.1=<hostname 1>:2888:3888`

`server.2=<hostname 2>:2888:3888`

`server.3=<hostname 3>:2888:3888`

`server.4=<hostname 4>:2888:3888`

Définition des variables d'environnement à mettre dans `bin/zkEnv.sh`

`export ZK_HOME=~/.zookeeper/`

`export PATH=$PATH:$ZK_HOME/bin:$ZK_HOME/conf`

`export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/`

Création du fichier myid : `echo <no de serveur> > ~/.zookeeper/data/myid`

Démarrer zookeeper : `bin/zkServer.sh start`

Connexion à zookeeper : `bin/zkCli.sh` comme client

Lister les znodes : `ls /`

Créer un znode : `create /workers ""` ("" signifie qu'on ne veut pas insérer de données)

Supprimer un znode : `delete /orkers`

Quitter le client : `quit`

Vérifier son statut : `echo srvr | nc localhost 2181 | grep Mode`

Arrêter Zookeeper : `bin/zkServer.sh stop`



### 10.3. Installation de HBASE

Téléchargement : wget <https://dlcdn.apache.org/hbase/2.4.6/hbase-2.4.6-bin.tar.gz>

Dans ~/.bashrc :

```
Export HBASE_HOME=/home/ubuntu/hbase
Export HBASE_CONF=$HBASE_HOME/conf
Export HBASE_WEBAPPS=$HBASE_HOME/hbase-webapps
Export PATH=$PATH:$HBASE_HOME/bin
```

Dans *conf/hbase-env.sh*, précision du JAVA HOME

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
export HBASE_REGIONSERVERS=/home/ubuntu/hbase/conf/regionserver
```

Dans *conf/hbase-site.xml* :

```
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://<hadoop-master>:9000/hbase</value>
</property>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
<property>
<name>hbase.zookeeper.quorum</name>
<value>hostname1, hostname2, hostname3...</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>hbase.zookeeper.property.clientPort</name>
<value>2181</value>
</property>
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/home/ubuntu/hbase/zookeeper</value>
</property>
</configuration>
```

Dans *conf/regionserver* : liste des hostnames des serveurs

Démarrer HBASE : *bin/start-hbase.sh*

Connexion à l'interface hbase : *./bin/hbase shell*

Arrêt de HBASE : *./bin/stop-hbase.sh*



## 10.4. Installation de HIVE

Téléchargement, décompression et renommage du dossier : `wget https://dldcn.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz && tar -zxf apache-hive-3.1.2-bin.tar.gz && rm -rf apache-hive-3.1.2-bin.tar.gz && mv apache-hive-3.1.2-bin hive`

Vérification des chemins dans ~/.bashrc :

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
export PATH=${JAVA_HOME}/bin:${PATH}
export HIVE_HOME=/home/ubuntu/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
```

Edition de conf/hive-env.sh :

```
cp hive-env.sh.template hive-env.sh
export HADOOP_HOME=/home/ubuntu/hadoop
```

Edition de bin/hive-config.sh :

```
export HADOOP_HOME=/home/ubuntu/hadoop
```

Initialiser la Derby database en rentrant dans le terminal la commande suivante : `$HIVE_HOME/bin/schematool -dbType derby -initSchema`

Installation de Derby : `wget https://dldcn.apache.org/db/derby/db-derby-10.14.2.0/db-derby-10.14.2.0-bin.tar.gz && tar -zxf db-derby-10.14.2.0-bin.tar.gz && mv db-derby-10.14.2.0-bin derby`

remplissage du .bashrc :

```
export DERBY_HOME=/home/ubuntu/derby
export PATH=$PATH:$DERBY_HOME/bin
export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

Création d'un dossier data dans derby : `mkdir -p ~/derby/data`

Configurer le métastore de Hive : `cd $HIVE_HOME/conf && cp hive-default.xml.template hive-site.xml && nano template hive-site.xml`

Créer un fichier jpox.properties dans lequel insérer les lignes suivantes :

```
javax.jdo.PersistenceManagerFactoryClass =
org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver
```

```
javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore_db;create = true
javax.jdo.option.ConnectionUserName = APP
javax.jdo.option.ConnectionPassword = mine
```

Lancer hadoop et créer dans hdfs un dossier /tmp et /user/hive/warehouse : `cd ~/hadoop && bin/hadoop fs -mkdir -p /tmp && bin/hadoop fs -mkdir -p /user/hive/warehouse`

Attribuer les droits à ces dossiers : `bin/hadoop fs -chmod g+w /tmp && bin/hadoop fs -chmod g+w /user/hive/warehouse`

Vérifier l'installation de hive : `cd ~/hive && bin/hive`

Test : lancement du terminal hive et création d'une table :

Lancer la commande (n'importe où) : `hive`

>create table product(product int, pname string, price float)

>row format delimited

>fields terminated by ',';

Normalement le terminal renvoie « OK ».

Puis taper : `describe product ;`

## 10.5. Installation de SPark



**Téléchargement et installation** : `wget https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz && rm -rf spark-3.2.0-bin-hadoop3.2.tgz`

Dans `.bashrc` :

```
export SPARK_HOME=/home/ubuntu/spark
```

```
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

**Créer spark-env.sh dans spark/conf** : `cp spark-env.sh.template spark-env.sh`

**Tester spark** : dans spark, taper `bin/spark-shell`

**Tester spark via url** : `http://ip-address:4040/`

**Installer pySpark** : `sudo apt install pip && pip install pyspark`

Dans le fichier `spark-env.sh` :

Ajout du `JAVA_HOME`\*\*

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
```

Ajout de l'adresse IP du master\*\*

```
export SPARK_MASTER='192.168.3.32'
```

Création du fichier slaves depuis `~/spark/conf/slaves`

```
# Ajouter les lignes suivantes au fichier slave
```

```
tp-hadoop-9 # master
```



```
tp-hadoop-10 # slave01
tp-hadoop-11 # slave02
tp-hadoop-12 # slave03
```

Lancement de Spark:

```
~spark/sbin/start-all.sh # pour lancer Spark
~spark/sbin/stop-all.sh # pour arrêter Spark
```

Ouverture du shell : *bin /spark-shell*

Lancer pySpark : pyspark

Programme Wordcount PySpark\*\*

**Creation** de ~/spark/WC\_spark/WC\_pyspark.py

Lancement du programme: bin/spark-submit ./WC\_pyspark/WC\_pyspark.py

```
bin/spark-submit ./WC_pyspark/WC_pyspark_file_arg.py ../hadoop/data/file02
```