Rapport INF 729 — Hadoop

Nicolas GALLAY, Jia LIAO, Urian MAILLARD, Philéas SAMIR, Gwladys SANCHEZ
MS BGD 21-22
Télécom Paris

***Résumé*—Nous avons travaillé sur une grappe de machines virtuelles, sur laquelle nous avons installé plusieurs logiciels open source d'Apache : Hadoop, HDFS, YARN, Zookeeper, HBase, Hive et Spark. Ces différents logiciels nous ont permis de distribuer le stockage et l'exécution d'applications sur la grappe. Nous présentons l'installation de chaque logiciel et ses défis propres, puis nous montrons l'intérêt de la distribution en stockant et en traitant un volume de données qui dépasse la capacité de stockage d'une machine seule.**

I. Introduction

L'écosystème distribué émerge au début des années 2000 du fait des besoins croissants des géants du Web de l'époque. Le *Google File System* [6] permet de passer à l'échelle le stockage, et le paradigme *MapReduce* [5] permet de passer à l'échelle l'exécution de tâches, sur des machines nombreuses mais peu performantes et en tolérant les pannes. Le *Hadoop Distributed File System* [9] formalise l'utilisation de MapReduce pour le stockage de petabytes de données de Yahoo. *Yet Another Resource Negotiator* (YARN) [11] permet de gérer plus efficacement les ressources des grappes. *ZooKeeper* [7] permet de distribuer des données d'application en maximisant la disponibilité et la coordination (désignation dynamique d'une machine leader, résilience aux partitions réseau). *HBase* est un système de base de données non-relationnel, basé sur Google Bigtable [3], qui repose sur un stockage distribué (HDFS). *Hive* [2] est un logiciel qui permet des interactions SQL-like avec des données distribuées (Hive traduit en jobs MapReduce les requêtes envoyées par le client). Enfin, *Spark* [14] [13] permet de dépasser le paradigme MapReduce en retirant certaines contraintes de linéarité. L'intégralité de ces logiciels est disponible en open source, donc gratuitement, grâce à l'Apache Software Foundation.
Afin de montrer l'intérêt de la distribution, mais aussi les défis liés à celle-ci, nous avons installé ces logiciels sur 5 machines virtuelles :

- machine maîtresse :
  - `tp-hadoop-2`
- machines slave :
  - `tp-hadoop-1`
  - `tp-hadoop-3`
  - `tp-hadoop-4`
  - `tp-hadoop-6`

Ces installations nous ont permis d'effectuer un WordCount sur un fichier trop volumineux pour être stocké ou traité sur une machine seule parmi celles de la grappe.

II. HDFS

A. Objectif

HDFS (Hadoop Distributed File System) est un système de fichier distribué qui a été conçu pour stocker de très gros volumes de données. Développé par Doug Cutting et Mike Cafarella à partir du Google FS, il est écrit en Java et permet d'éviter une architecture physique de stockage en manipulant un système de fichiers distribué comme un disque dur unique. Son installation est l'élément de base de notre cluster Hadoop. Cela va nous aider à réaliser notre calcul distribué en répartissant de manière optimale le stockage entre nos 5 machines.

B. Configuration

La première étape consistait à accéder aux machines de l'école. Pour ce faire, nous avons fait un `ssh-keygen` dans le terminal afin de générer une paire de clés publique/-privée et nous sommes allés récupérer la clé publique en utilisant la commande `cat ~/.ssh/id_rsa.pub`. Nous avons ensuite échangé nos clés publiques avec les machines de l'école via le canal « Mattermost ». Une fois la clé publique de la machine de TP récupérée, nous l'avons enregistrée dans les clés autorisées (`~/.ssh/authorized_keys`). La connexion SSH établie, il nous suffit maintenant de nous connecter au bridge puis à la machine souhaitée en utilisant :

```
ssh ubuntu@ip
ssh tp-hadoop-(num_machine_souhaitée)
```

La connexion SSH établie, il convient d'installer la dernière version de Java (Java 8) ainsi que d'autres pré-requis :

```
sudo apt install openjdk-8-jre-headless
sudo apt-get install ssh
sudo apt-get install pdsh
```

Une fois les prérequis effectués, il convenait d'installer Hadoop sur chaque machine. Hadoop se trouve gratuitement sur internet en format `tar.gz`. Nous téléchargeons le fichier à l'aide de la fonction `wget` pour ensuite décompresser le fichier en utilisant `tar -zxvf`. HDFS étant écrit en Java, il est important d'aller dans le dossier Hadoop nouvellement créé pour modifier le fichier `hadoop-env.sh` en y insérant le lien pour l'installation de Java : `export JAVA_HOME=/usr/java/latest`. Afin de s'approprier au mieux l'outil, nous avons d'abord effectué des tests en « Standalone Mode» (local mode). Pour ensuite

tester le « Pseudo Distributed Mode » (Single Node Cluster) qui utilise seulement un single node en simulant une grappe où toutes les procédures vont tourner de manière indépendante les unes des autres. Enfin nous sommes passés au mode « Fully Distributed » (Multi-Node Cluster) où l'une des machines agit en « Master » (Namenode et Resource Manager) qui va donner des instructions à des « Slaves » (Data Node et Node Manager). Ces étapes incluent l'installation de YARN que nous verrons par la suite.

C. Problèmes rencontrés

Nous avons rencontré des problèmes lors du lancement de HDFS où la commande `sbin/start-dfs.sh` entrainait le message d'erreur `error : rcmd :socket :Permission denied`. Nous avons finalement réussi à contourner le problème en utilisant les commandes suivantes :

```
sudo -i
echo "ssh" > /etc/pdsh/rend-default
exit
```

Nous présentons les résultats de l'installation dans la partie YARN (HDFS et YARN sont packagés ensemble). Voir III-D.

III. YARN

A. Objectif

On souhaite installer YARN (Yet Another Resource Negotiator), ce qui nous permettra d'allouer les ressources du système aux différentes applications exécutées dans notre cluster ainsi que de planifier l'exécution des tâches sur les machines de la grappe.

B. Configuration

Pour utiliser YARN, la première étape est de configurer les rôles des différentes machines. Dans le répertoire `~/hadoop-3.3.1/etc/hadoop`, on insère de nouvelles configurations sur les fichiers :

- `core-site.xml` afin de nommer la machine 2 en tant que « master », voir Annexe X-A
- `workers` afin de définir les machines qui seront des « slaves », voir Annexe X-B
- `hdfs-site.xml` afin d'installer un chemin pour HDFS, voir Annexe X-C
- `mapred-site.xml` afin de définir YARN comme un « planificateur de tâches », voir Annexe X-D
- `yarn-site.xml`, voir Annexe X-E

Il convient aussi de définir des variables d'environnement en modifiant les fichiers cachés au niveau du « home » :

- `.profile`, voir Annexe X-F
- `.bashrc`, voir Annexe X-G

C. Problèmes rencontrés

Lors du passage en « Fully Distributed » les configurations étant nombreuses, il a été difficile de configurer les machines de la même manière sans générer d'erreurs. C'est pourquoi il a été convenu de changer notre méthodologie en appliquant les modifications sur une seule machine de la grappe pour ensuite « pusher » une copie de celle-ci sur les 4 machines restantes du cluster. Cette approche a permis un gain de temps dans nos échanges tout en s'assurant que toutes les machines soient bien configurées de la même manière.

D. Résultats obtenus

Afin de tester YARN et HDFS, nous avons effectué un Mapreduce à partir des exemples présents lors de l'installation. Nous avons ainsi créé un fichier « input » et nous avons mis les résultats du test dans un fichier « output ». Le Mapreduce a fonctionné correctement.

```
bin/hdfs dfs -mkdir input
bin/hdfs dfs -put etc/hadoop/*.xml input
bin/hadoop jar share/hadoop/mapreduce/
hadoop-mapreduce-examples-3.3.1.jar
grep input output 'dfs[a-z.]+'
bin/hdfs dfs -cat output/*
```

IV. ZooKeeper

A. Objectif

Zookeeper est un outil de gestion de configuration pour systèmes distribués, il garde la grappe et sa configuration en ordre.

B. Configuration

1) Télécharger et dézipper la dernière version stable (`apache-zookeeper-3.6.3-bin.tar.gz`, il ne faut pas prendre `apache-zookeeper-3.6.3.tar.gz`)
2) Copier sur toutes les machines
3) Modifier les fichiers de configuration sur le master :
   a. `zoo.cfg` (sous `/home/ubuntu/apache-zookeeper-3.6.3-bin/conf/`)
      i. copier le fichier `zoo\_sample.cfg` vers `zoo.cfg`
      ii. ajouter :
      ```
      dataLogDir=/home/hadoop/
      zookeeper-3.6.3/logs
      server.1=tp-hadoop-2:2888:3888
      server.2=tp-hadoop-1:2888:3888
      server.3=tp-hadoop-3:2888:3888
      server.4=tp-hadoop-4:2888:3888
      server.5=tp-hadoop-6:2888:3888
      ```
      iii. modifier la destination du *datadir* pour pointer vers `$ZOOKEEPER_HOME/data` (préalablement créé)
   b. `.bashrc` (sous `/home/ubuntu/`)
      i. ajouter (pour pointer vers le dossier zookeeper) :
      ```
      export ZOOKEEPER_HOME=/home/ubuntu/
      apache-zookeeper-3.6.3-bin
      export PATH=$PATH:
      $ZOOKEEPER_HOME/bin
      ```

ii. `source /home/ubuntu/.bashrc` (pour rendre la variable de l'environnement valide)

4) Pusher le dossier `apache-zookeeper-3.6.3-bin` et le fichier `.bashrc` sur toutes les machines
5) Créer un fichier `myid` sur toutes les machines, qui lui indique son id :

```
ssh tp-hadoop-2 echo 1 >
$ZOOKEEPER_HOME/data/myid
ssh tp-hadoop-1 echo 2 >
$ZOOKEEPER_HOME/data/myid
ssh tp-hadoop-3 echo 3 >
$ZOOKEEPER_HOME/data/myid
ssh tp-hadoop-4 echo 4 >
$ZOOKEEPER_HOME/data/myid
ssh tp-hadoop-6 echo 5 >
$ZOOKEEPER_HOME/data/myid
```

6) Commandes utilisées (sous `/home/ubuntu/apache-zookeeper-3.6.3-bin/bin/`) :
   - `zkServer.sh start` : démarrer le server Zookeeper, à faire sur au moins 3 machines.
   - `zkServer.sh status` : mode = leader/follower signifie OK ; standby signifie NOK.
   - `zkServer.sh stop` : arrêter le server Zookeeper.
   - `zkCli.sh` : démarrer le client Zookeeper :
     - création de znodes :
       - `create /FirstZnode "MyFirstZookeeperApp"`
       - `create -e /SecondZnode "Data"` (ephemereal node)
     - récupération de données : `get /FirstZnode`

C. Problèmes rencontrés

- Quand il y a une seule machine avec un zkServer sur les 5, le zkCli ne se lance pas : C'est normal. Parce qu'il est en minorité, i.e. il sait qu'il y a 5 machines et il ne fait rien s'il n'est pas connecté à au moins 2 autres machines. Il faut au moins 3 machines qui tournent et connectées pour lancer le client.
- Il n'y a plus de Main Zookeeper quand on arrête (`zkServer.sh stop`) le main (`tp-hadoop-2`) : C'est normal. Parce qu'il n'a pas encore eu besoin de désigner un nouveau Main.

D. Résultats obtenus

La commande `jps` (Java Process Status) permet d'avoir un *status* de la grappe Hadoop, voir la Table I pour les résultats après l'installation de HDFS, YARN, Zookeeper et Hbase.

Quand nous démarrons Zookeeper server :

1) Nous visualisons bien ZookeeperMain sur le Master et QuorumPeerMain sur les Workers.
2) `zkServer.sh status` montre mode = leader sur le Master et mode = follower sur les workers.

	Master	Worker
HDFS	NameNode, Secondary NameNode	DataNode
YARN	ResourceManager	NodeManager
Zookeeper	ZookeeperMain, QuorumPeerMain	QuorumPeerMain
HBase	HMaster, HQuorumPeer	HRegionServer, HQuorumPeer

TABLE I
PROCESSUS JAVA PAR TYPE DE MACHINE ET PAR LOGICIEL DE DISTRIBUTION

3) le client Zookeeper se lance correctement.

V. HBASE

A. Objectif

Jusqu'ici, l'utilisation de Hadoop reposait sur le système de stockage de données HDFS, précédemment décrit. Toutefois, celui-ci présente certaines limites, et n'est pas forcément adapté à tous les types de fichiers ou de projets. En particulier, dans le cas de travaux qui nécessitent un débit important, avec des requêtes fréquentes, ou qui concernent des données aléatoirement réparties dans toute la grappe, le système HDFS ne permet pas d'atteindre de bonnes performances. Afin d'optimiser ce type de calculs, HBase a été ajouté aux "briques" disponibles pour Hadoop.

Application directe du système BigTable développé par Google en 2006, il s'agit d'un outil qui se repose sur HDFS, mais qui propose une réorganisation des données selon un modèle de datastore orienté "famille de colonne", similaire aux bases de données relationnelles et au langage (no)SQL. HBase permet ainsi de facilier l'accès aux données lorsqu'elles sont de taille importante, et d'optimiser les requêtes effectuées.

B. Configuration

Afin d'installer HBase dans notre grappe, nous avons tout d'abord procédé, sur la machine maîtresse tp-hadoop-2, au téléchargement et dézippage de l'archive relative à la version v2.4.6 de HBase, par le biais de la commande `wget https://dlcdn.apache.org/hbase/2.4.6/hbase-2.4.6-bin.tar.gz`. Ensuite, trois fichiers de configuration ont été modifiés comme suit, dans le dossier `~/hbase-2.4.6/conf/` :

- `hbase-env.sh` : la ligne du *path* vers JAVA_HOME a été décommentée et remplacée par la commande (`export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/`). Ceci permet de s'assurer que le programme sache à quel endroit chercher les outils Java dont il a besoin sur la machine. Le *path* vers JAVA_HOME a été modifié et ajouté en variable d'environnement (`~/.bashrc`), mais cette ligne a été modifiée quand même par précaution, pour éviter les erreurs en cas de modification ultérieure des fichiers de configuration globaux.
- `regionservers` : ce fichier, qui contenait uniquement `localhost`, a été modifié de façon à contenir sur chaque ligne, le nom des machines travailleuses, en l'occurrence `tp-hadoop-1`, `tp-hadoop-3`, `tp-hadoop-4`, `tp-hadoop-6`.

— `backup-masters` : ce fichier a été créé, et la ligne `tp-hadoop-1` y a été inscrite, afin de désigner la machine n°1 comme "backup master" de la grappe.

A ce stade, en tentant de démarrer HBase avec la commande `start-hbase.sh`, l'erreur suivante apparaissaît : `SLF4J: Class path contains multiple SLF4J bindings`, qui indiquait l'existence d'un conflit entre le logging de Hadoop à SLF4J et celui de HBase. Suivant les instructions trouvées sur un forum en ligne, une solution a été apportée à ce problème en supprimant un script java de de type `slf4j-log4j`, situé à au chemin `hbase/lib/client-facing-thirdparty/slf4j` [8].

Afin d'éviter que HBase ne lance sa propre instance de Zookeeper, mais se base bien sur celle déjà installée sur la grappe, nous avons ensuite modifié les paramètres dans le fichier de configuration `hbase-site.xml`, en y ajoutant les lignes suivantes :

```
<configuration>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
</configuration>
```

ainsi que le fichier `/hbase-env.sh`, dans lequel nous avons décommenté et modifié la ligne `export HBASE_MANAGES_ZK=false`.

D'autres modifications ont également été ajoutées à ce fichier, en définissant les couples <name>-<value> suivants dans trois autres propriétés de configuration :

— Afin d'indiquer à HBase le port à utiliser, identique à celui utilisé par Hadoop, déjà défini dans `hadoop/core-site.xml` :
```
<name>hbase.rootdir</name>
<value>hdfs://localhost:9000/hbase
   </value>
```

— Afin d'indiquer à HBase les adresses des machines constituant la grappe, dans l'ordre Master - Travailleuses :
```
<name>hbase.zookeeper.quorum</name>
<value>tp-hadoop-2, tp-hadoop1,
tp-hadoop3, tp-hadoop4, tp-hadoop6
   </value>
```

— Afin d'indiquer à HBase le *path* vers l'instance de Zookeeper déjà installée sur la grappe :
```
<name>hbase.zookeeper.property.dataDir
</name>
<value>/home/ubuntu/apache-Zookeeper
-3.6.3-bon</value>
```

Après ces modifications, le Master HBase a bien pu être lancé, par le biais de la commande `start-hbase.sh`, et la configuration (c'est-à-dire tout le dossier `~/hbase-2.4.6/`) a été *pushée* par `scp` sur les autres machines de la grappe.

C. Problèmes rencontrés

La première difficulté à laquelle nous avons été confrontés en utilisant HBase, a été de procéder à son arrêt, par exemple lorsque nous voulions tout stopper pour relancer un démarrage global de tous les outils de Hadoop. La commande `stop-hbase.sh` semblait en effet inopérante, et lorsque nous vérifiions les processus en cours sur la grappe par le biais de la commande `jps`, les processus HBase apparaissaient toujours. La solution qui nous a ensuite été apportée consistait à lancer la commande `bin/hbase-daemons.sh stop regionserver`, qui avait pour résultat d'arrêter les processus de HBase sur toutes les machines de la grappe.

La deuxième difficulté rencontrée concerne l'utilisation de HBase proprement dite. Une fois lancée sur toutes les machines, et ce sans erreur, nous avons tenté de créer une table à partir du shell HBase. Nous recevions alors systématiquement le message d'erreur `Server is not running yet`. Nous avons tenté de copier tous les fichiers sous `~/hadoop-3.3.1/share/hadoop/tools/lib/` vers `~/hbase/lib/` et de supprimer les fichiers redondants (avec une version plus ancienne). Le contenu du répertoire `lib` a été sauvegardé dans le répertoire `lib_backup`. Mais cela n'a pas eu l'effet escompté, et l'erreur apparaissaît toujours après modification.

Finalement, nous ne sommes pas parvenus à faire fonctionner HBase sur notre grappe.

VI. Hive

A. Objectif

Initialement développé par Facebook en 2007, le logiciel Hive permet de simplifier l'utilisation de MapReduce, pour lequel les codes sont souvent difficiles à écrire (gestion de la phase de mapping, de la phase de réduction...). Hive, quant à lui, fournit une abstraction SQL (HiveQL), qui permet d'écrire des requêtes et d'en obtenir les réponses de façon similaire au langage SQL, bien que le calcul passe quand même par MapReduce. Hive ne constitue pas une base de données, mais uniquement cette abstraction langagière destinée à simplifier les travaux MapReduce. En revanche, il peut s'interfacer avec HBase si on le souhaite.

B. Configuration

Nous avons procédé au téléchargement et dézippage du fichier `apache-hive-3.1.1-bin`, puis avons ajouté le chemin `HIVE_HOME` dans le fichier `PATH` tel qu'indiqué dans la documentation officielle de Apache.

C. Mise en oeuvre

N'étant pas parvenus à faire fonctionner HBase, nous n'avons pas testé le fonctionnement de Hive, et sommes plutôt directement passés à l'étape suivante : l'installation de Spark.

VII. Spark

A. Objectif

Spark est un Framework de calcul distribué créé en 2009 faisant aujourd'hui partie des projets de la fondation Apache. Il permet de faciliter la mise en place de grappes et l'exécution de calculs distribués à grande échelle en fournissant une interface pour la programmation. Il permet l'utilisation de langages tel que Scala, Java, Python, et R. Spark n'utilise pas Map Reduce et s'en différencie en exécutant les opérations directement en mémoire lui permettant d'être plus rapide.

B. Configuration

Nous avons eu recours à [10], [12]. La procédure d'installation du Framework Spark est simple. Il faut tout d'abord télécharger et décompresser l'archive contenant Spark sur une des machines de la grappe. Spark ne dispose pas de système de gestion de fichier qui lui est propre. Il est donc nécessaire de lui en fournir un en l'intégrant à Hadoop. Spark propose des versions intégrant les librairies Hadoop lui permettant notamment d'accéder et d'utiliser le système de fichier HDFS. Notre grappe possédant déjà un Hadoop paramétré, nous avons choisi la version "without hadoop".
Téléchargement de Spark 3.2.0 (sans Hadoop) depuis le site web officiel d'Apache Spark :
`wget https://downloads.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-without-hadoop.tgz`
Création du répertoire d'installation Spark :
`mkdir /home/ubuntu/spark-3.2.0`
Décompression des fichiers Spark dans ce répertoire :
`tar xvf spark-3.2.0-bin-without-hadoop.tgz -C /home/ubuntu/spark-3.2.0`
Nous définissons dans le fichier `~/.bashrc` la variable d'environnement de Spark (*SPARK_HOME*) ainsi que son PATH :

```
export SPARK_HOME=/home/ubuntu/spark-3.2.0
export PATH=$SPARK_HOME/bin:$PATH
```

Enfin, nous modifions le fichier de configuration (`spark-env.sh`) de Spark en ajoutant la ligne suivante :

```
ubuntu@tp-hadoop-1::~/spark-3.2.0/conf$
nano spark-env.sh
export SPARK_DIST_CLASSPATH=$(/home/ubuntu/
hadoop-3.3.1/bin/hadoop classpath)
```

Ceci nous permet de connecter Spark au Hadoop installé précédemment.

Une fois les fichiers et la configuration copiés sur toutes les machines de la grappe, Spark est prêt à être utilisé. La commande `spark-shell` permet alors de lancer un shell spark en scala mais il est possible, comme dit précédemment, d'utiliser d'autres langages tel que python en ouvrant un shell PySark `./bin/pyspark`.

C. Problèmes rencontrés

L'installation de Spark étant relativement simple, nous n'avons pas rencontré de problème particulier.

D. Résultats obtenus

Afin de s'assurer du bon fonctionnement de Spark, nous avons pu tester la création d'un RDD ainsi qu'un calcul simple en scala par l'intermédiaire du Shell Spark. Après avoir mis un fichier text dans notre HDFS, nous avons lancé un "count words" et récupérer en fichier de sortie contenant les occurences de tous les mots.

```
spark-shell
scala> val inputfile =
sc.textFile("input.txt")
scala> val counts = inputfile
.flatMap(line => line.split(" "))
.map(word => (word, 1)).reduceByKey(_+_);
scala> counts.toDebugString
scala> counts.cache()
scala> counts.saveAsTextFile("output")
hadoop dfs -get output
```

Ce calcul nous a permis de verifier la connexion entre Spark et notre HDFS.

VIII. Utilisation de la grappe

A. MapReduce Wordcount

L'objectif de ce Wordcount était de "stress-tester" la grappe, c'est-à-dire d'évaluer la capacité de la grappe à stocker un volume de données important et à être capable d'en extraire de l'information. On entend par "volume de données important" un fichier ou une collection de fichiers qu'une machine seule de la grappe ne serait pas en mesure de stocker. Ces objectifs nous permettent de montrer l'intérêt pratique et concret de HDFS et de MapReduce.

Nous avons choisi les données Wikipedia comme input. Le volume de Wikipedia en anglais étant tel qu'on ne peut envisager de le stocker sur HDFS avec un facteur de réplication supérieur à 1 (ce qui pose des problèmes de fiabilité en cas de panne), nous avons choisi d'utiliser l'intégralité des pages Wikipedia en français au format XML. Les pages sont toutes regroupées dans un même fichier, d'une taille de 37.8 Go (stocké sur les 4 worker nodes avec un facteur de réplication spécialement fixé à 2 soit 75.6 Go sur les disques — le facteur 3 initialement fixé n'empêchait pas le stockage, mais ralentissait trop fortement les machines workers pour toute autre opération, y compris la connexion).

On lance le Wordcount présent dans les exemples fournis avec l'installation d'HDFS sur ce fichier. Le Wordcount a fonctionné et a renvoyé un dossier `output` qui contient bien le fichier de résultats.

Le fichier de résultats a une taille de 5.3 Go, ce qui donne 16 Go sur disque du fait du facteur de réplication de 3 (on rappelle que seul le facteur de réplication du fichier d'input a été manuellement changé). Ce résultat paraît énorme et s'explique notamment par le fait que le Wordcount sépare

les mots sur les espaces, retours de lignes et tabulations ; or, le fichier d'entrée est un fichier structuré XML. Il faudrait définir une application Mapreduce de pré-traitement du fichier XML qui nous permettrait d'extraire les informations qui nous intéressent, avant de compter les mots.

Le fichier de résultats contient les mots présents dans le fichier d'entrée et le nombre d'occurrences, et est trié par clef, c'est-à-dire par ordre alphabétique croissant. Cela se vérifie notamment grâce aux commandes `-head` et `-tail` du dfs (on regrette cependant l'absence de l'option `-n` dans les implémentations du dfs), ou `-cat` puis un pipe et un `head -n` classique (ou même un `tail -n` si on est assez patient). On recommande de regarder la tail du résultat afin de voir apparaître tous les emojis ou les combinaisons d'emojis présents sur le Wikipedia francophone (voir par exemple 1).

B. TFIDF et KMeans

Les matrices TF.IDF sont aujourd'hui largement utilisées pour la fouille de textes et la mesure de similarité dans des systèmes de recommandation [1]. Dans ce cadre, nous avons tenté de calculer une matrice TF.IDF de façon distribuée sur le corpus Wikipedia et sur des fichiers CommonCrawl [4] téléchargés pour l'occasion.

Nous avons rédigé un script Python qui permet de parser un fichier XML, de calculer la matrice (Tokenisation, Term Frequency, Inverse Document Frequency puis produit TF x IDF), et enfin de clusteriser les documents grâce aux vecteurs TFIDF de chaque document. L'espace de représentation pour le clustering a un très grand nombre de dimensions (égal au nombre de mots distincts sur le wikipedia francophone, sachant que les fichiers d'entrée i.e. les pages ne sont pas nettoyées — il reste les balises, la ponctuation, etc). Un second script Python permet d'effectuer la même opération sur un dossier de fichiers CommonCrawl sur le DFS (volume des fichiers CC = 17.6 Go, nombre de fichiers = 45).

Le script Common Crawl fonctionne pour des fichiers .txt peu volumineux, mais ne passe pas à l'échelle (problèmes de mémoire, même si on limite à seulement quelques fichiers).

IX. Conclusions

Dans ce rapport, nous détaillons les étapes que nous avons suivies pour installer plusieurs outils de distribution disponibles en open source sur une grappe de machines virtuelles. Nous présentons les difficultés que nous avons rencontrées et les solutions que nous avons apportées pour les dépasser. Ce module nous a permis de nous familiariser avec l'invite de commandes, le langage bash, les connexions ssh, et les défis liés à la synchronisation de machines et à l'installation simultanée d'outils et de logiciels.

Notre Wordcount final permet de montrer l'intérêt de la distribution puisque le fichier d'entrée dépasse les capacités de stockage d'une machine de la grappe seule, et dépasse largement les capacités de mémoire vive des machines personnelles disponibles sur le marché. Cette opération aurait donc été difficile et lente sans distribution (mais pas impossible !).

Nous pourrions aller plus loin en poursuivant les axes suivants :

- faire fonctionner HBase et Hive,
- écrire un script pour nettoyer le fichier XML de Wikipedia de façon distribuée,
- on pourrait traiter les métadonnées des pages Wikipedia comme les différentes colonnes d'une base HBase, et effectuer des requêtes et agrégations sur cette base,
- on suppose que les erreurs mémoire de nos scripts TF.IDF viennent de notre implémentation : il faudrait donc travailler sur un script qui ne saturerait pas la mémoire pour réussir à aller au bout de l'opération.

Références

[1] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems : a literature survey. *International Journal on Digital Libraries*, 17(4) :305–338, 2016.

[2] Jesús Camacho-Rodríguez, Ashutosh Chauhan, Alan Gates, Eugene Koifman, Owen O'Malley, Vineet Garg, Zoltan Haindrich, Sergey Shelukhin, Prasanth Jayachandran, Siddharth Seth, Deepak Jaiswal, Slim Bouguerra, Nishant Bangarwa, Sankar Hariappan, Anishek Agarwal, Jason Dere, Daniel Dai, Thejas Nair, Nita Dembla, Gopal Vijayaraghavan, and Günther Hagleitner. Apache hive : From mapreduce to enterprise-grade big data warehousing. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1773–1786, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable : A distributed storage system for structured data. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 205–218, 2006.

[4] Common Crawl. Common crawl. https://commoncrawl.org/. Accessed : 11-12-2021.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : Simplified data processing on large clusters. In *OSDI'04 : Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 20–43, Bolton Landing, NY, 2003.

[7] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper : Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, page 11, USA, 2010. USENIX Association.

[8] james. slf4j. https://stackoverflow.com/questions/9393499/hbase-0-92-warnings-about-slf4j-bindings. Accessed : 11-12-2021.

[9] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[10] tutorialspoint. Apache spark core programming. https://www.tutorialspoint.com/apache_spark/apache_spark_core_programming.htm. Accessed : 06-12-2021.

[11] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn : Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, New York, NY, USA, 2013. Association for Computing Machinery.

[12] Tom White. *Hadoop : The Definitive Guide*. O'Reilly, Beijing, 4 edition, 2015.

[13] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets : A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 2, USA, 2012. USENIX Association.

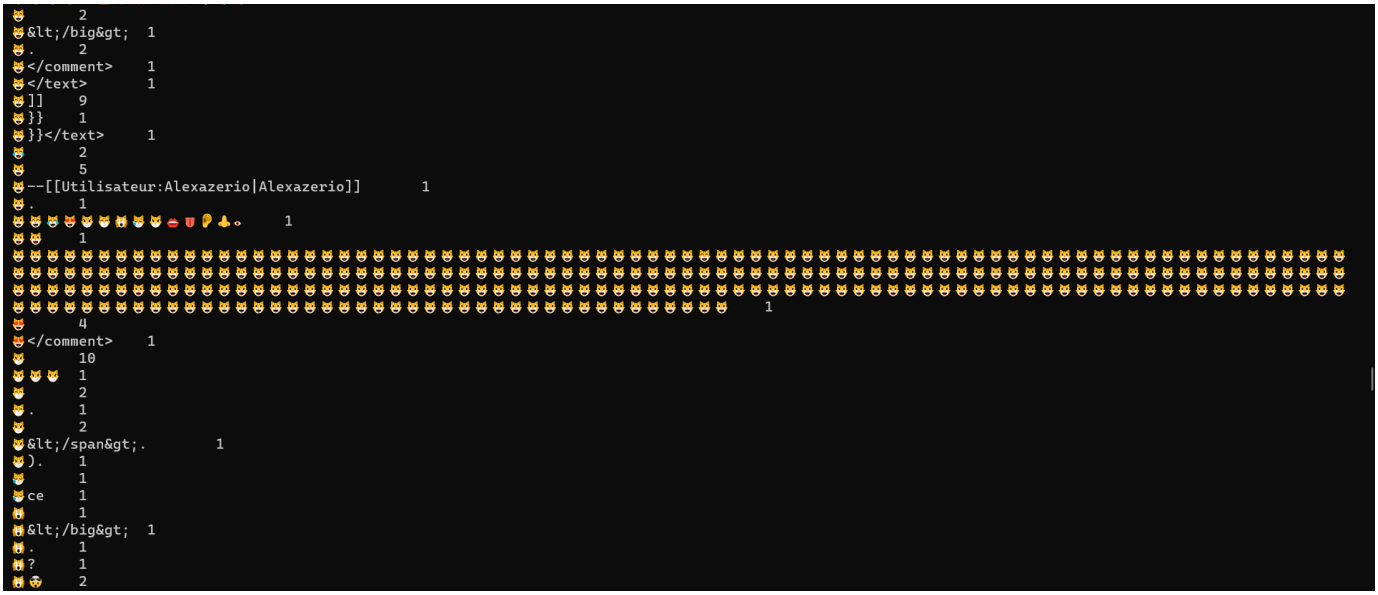

FIGURE 1. Incidence de l'émoji chat sur le Wikipedia francophone. Cet exemple met aussi en évidence les problèmes liés à l'absence de pré-traitement du XML.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark : Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, USA, 2010. USENIX Association.

X. ANNEXES

A. Annexe 1 - core-site.xml

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>
        hdfs://tp-hadoop-2:9000
        </value>
    </property>
</configuration>
```

B. Annexe 2 - workers

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>
        hdfs://tp-hadoop-2:9000
        </value>
    </property>
</configuration>
cat workers
tp-hadoop-1
tp-hadoop-3
tp-hadoop-4
tp-hadoop-6
```

C. Annexe 3 - hdfs-site.xml

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
        </property>
    <property>
            <name>
            dfs.namenode.name.dir
            </name>
            <value>
            file:///home/ubuntu/
            hadoop-3.3.1/
            data/namenode
            </value>
    </property>
    <property>
            <name>
            dfs.datanode.data.dir
            </name>
            <value>
            file:///home/ubuntu/
            hadoop-3.3.1/
            data/datanode
            </value>
    </property>
</configuration>
```

D. Annexe 4 - mapred-site.xml

```
<configuration>
    <property>
            <name>
            mapreduce.framework.name
            </name>
            <value>yarn</value>
    </property>
```

```
    <property>
            <name>
            yarn.app.mapreduce.am.env
            </name>
            <value>
            HADOOP_MAPRED_HOME=$HADOOP_HOME
            </value>
    </property>
    <property>
            <name>mapreduce.map.env</name>
            <value>
            HADOOP_MAPRED_HOME=$HADOOP_HOME
            </value>
    </property>
    <property>
            <name>mapreduce.reduce.env</name>
            <value>
            HADOOP_MAPRED_HOME=$HADOOP_HOME
            </value>
    </property>
</configuration>
```

E. Annexe 5 - yarn-site.xml

```
<configuration>
    <property>
            <name>
            yarn.acl.enable
            </name>
            <value>0</value>
    </property>

    <property>
            <name>
            yarn.
            resourcemanager.
            hostname
            </name>
            <value>tp-hadoop-2</value>
    </property>

    <property>
            <name>
            yarn.
            nodemanager.
            aux-services
            </name>
            <value>
            mapreduce_shuffle
            </value>
    </property>
</configuration>
```

F. Annexe 6 - home - .profile

```
fi
PATH=/home/ubuntu/hadoop-3.3.1/bin:
/home/ubuntu/hadoop-3.3.1/sbin:$PATH
```

G. Annexe 7 - home - .bashrc

```
fi
export PDSH_RCMD_TYPE=ssh
#Hadoop
export HADOOP_HOME=~/hadoop-3.3.1/
export PATH=${PATH}:
${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin
```