École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2012 - 2013

Rapport Projet Option Web & Multimédia

Hadoop : Mise en application du concept MapReduce

Encadrant
Romain RAVEAUX
romain.raveaux@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Jérôme HEISSLER
jerome.heissler@etu.univ-tours.fr
François SENIS
francois.senis@etu.univ-tours.fr

DI5 2012 - 2013

Version du 26 avril 2013

Table des matières

Table des figures

Liste des tableaux

Introduction

Durant la 5ème année du cursus ingénieur en informatique à Polytech'Tours, nous avons eu pour objectif de réaliser un projet d'option Web & multimédia. Ce projet a été réalisés en binôme. Le but de ce projet est de permettre aux étudiants d'améliorer leurs compétences en programmation, mais également de conduire un projet du début à la fin en respectant les contraintes imposées par le client qui, dans notre cas, est représenté par un professeur de l'école.

Présentation du projet

2.1 But du projet

Notre projet consiste à étudier le framework Hadoop développé par Apache. Dans un premier temps nous devions installer et prendre en main Hadoop. Hadoop dispose d'une implémentation complète du concept MapReduce. C'est un modèle de programmation parallèle facilitant le développement d'applications distribuées. Après avoir bien compris le fonctionnement d'un job MapReduce, l'objectif était de mettre en place un job simple. Puis, le but principal du projet était de mettre en place un comparateur de documents textuels qui s'appuiera sur la mesure de similarité Cosinus. Pour finir et si le temps nous le permet de le déployer sur une architecture distribuée.

2.2 Présentation d'Hadoop

Hadoop est un framework Open Source géré par la fondation Apache pour réaliser des traitements sur des gros volumes de données, de l'ordre de plusieurs petaoctets (soit plusieurs milliers de To). Il fait donc parti du domaine du Big Data. Hadoop est écrit en Java et de ce fait est exécuté dans une JVM. Malgré le problème de performance souvent associé à Java par rapport au C++, le choix du Java a été fait afin de pouvoir profiter de la communauté associée à Java (qui est la plus grande au monde), mais aussi du fait de ne pas avoir à être dépendant d'une architecture système étant donné qu'Hadoop est exécuté dans une JVM qui est disponible sur toutes les architecture Unix/Windows. De ce fait Hadoop est déployable sur n'importe quel machine.

Hadoop a été conçu par Doug Cutting en 2004. Egalement à l'origine du moteur Open Source Nutch (mais également de Lucene), Doug Cutting cherchait une solution pour accroître la taille de l'index de son moteur. Il eut l'idée de créer un framework de gestion de fichiers distribués qui est devenu Hadoop.
Par la suite Yahoo ! est devenu le principal contributeur. Il utilisait notamment l'infrastructure pour supporter son moteur de recherche historique. Comptant plus de 10 000 clusters Linux en 2008, il s'agissait d'une des premières architectures Hadoop digne de ce nom... avant que Yahoo ! ne décide de baser son moteur sur Microsoft Bing en 2009. Continuant à recourir à ce socle pour sa gestion de contenu Web et de ses annonces publicitaires, le groupe décide en 2011 de lancer une société (baptisée Hortonworks) avec pour objectif de proposer une offre de services autour d'Hadoop.

Tout le fonctionnement d'Hadoop repose sur une architecture distribuée. Lorsque l'on déploie un cluster de machine, on définit un serveur maitre aussi appelé namenode et un ensemble de machine esclave que l'on appelle datanode. Si l'on prend un exemple simple de processus étant exécuté sur Hadoop : le système de fichier HDFS. Tous les fichiers stockés sont répartis entre les datanode. Lorsqu'un client demande un fichier, il fait sa demande au namenode qui va interroger tous les datanode afin de savoir quel nœud stocke celui-ci. Le namenode renverra au client un lien vers le fichier sur le datanode. Grâce à ce système de fonctionnement, Hadoop est capable d'avoir un espace de stockage illimité tant que l'on ajoute des datanodes. Il en est de même pour les jobs MapReduce où les traitements sont répartis sur les datanodes afin d'être agrégés par la suite, mais nous reviendrons sur ce fonctionnement plus tard.

2.2.1 Installation et configuration d'Hadoop

L'installation a été testée sur une version Linux 12.04 LTS et une distribution mac OSX. L'installation sous mac OSX se fait via l'utilitaire HomeBrew et ne sera pas détaillée dans le rapport.

Installation de JAVA

Pour installer correctement Hadoop, il y a un certain nombre de prérequis. En effet, pour disposer du meilleur fonctionnement d'Hadoop, il est préférable que l'utilisateur dispose de la dernière version de java installée sur son système. Pour cela, il suffit de rentrer la commande suivante :

Listing 1 Installation de java 6

```
1 sudo apt-get install sun-java6-jdk && sudo update-javaalternatives -s java-6-sun
```

Toutefois, il est important de noter que la version 6 et la version minimum requise pour faire fonctionner Hadoop.

Pour connaitre la version Java qui est installée, il suffit d'entrer la commande suivante :

Listing 2 Connaitre version de java

```
1 java -version
```

Création d'un utilisateur

Le processus Hadoop sera par la suite exécuté par un utilisateur particulier qui sera entièrement dédié à l'utilisation de Hadoop. Pour cela, nous allons créer un utilisateur nommé hd_user appartenant au groupe hadoop sur chacune des machines exécutant Hadoop via les commandes suivantes :

Listing 3 Création de l'utilisateur Hadoop et du groupe Hadoop

```
1 sudo addgroup hadoop
2 sudo adduser --ingroup hadoop hd_user
```

Connexion SSH sans aucun mot de passe

Lorsque l'on déploie une architecture Hadoop, tous les noeuds du cluster communique entre eux via SSH. Cela permet par exemple au namenode de connaitre l'état de toutes les machines du cluster. Cependant lorsque l'on initie une connexion SSH un login et un mot de passe est requis pour établir la connexion. Pour cela nous allons créer une paire de clé RSA afin de pouvoir lancer la connexion sans demandé de mot de passe pour l'utilisateur hd_user. Avant toute chose, il est important de s'assurer que le service SSH est bien installé sur votre machine. Ensuite il faut générer une paire de clés RSA pour permettre une connexion sans mot de passe. Pour cela il vous suffit de rentrer la commande suivante :

Listing 4 création d'une pair de clé RSA

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

Ensuite il vous suffit de copier cette clé sur chacun des datanodes en rentrant les commandes suivantes :

Listing 5 transfère de la clé publique

```
ssh hd_user@Datanode1 mkdir -p .ssh
cat .ssh/id_rsa.pub | ssh hd_user@Datanode1 'cat >> .ssh/authorized_keys'
```

Téléchargement/Installation d'Hadoop

Téléchargez une version stable de Hadoop à partir de la page d'Apache Hadoop disponible à l'adresse suivantehttp://hadoop.apache.org/core/releases.html. En ce qui nous concerne, nous avons utilisé la version 1.1.2 d'Hadoop.

Décompressez le contenu de la distribution dans un endroit raisonnable, comme « /usr/local » par exemple (« /opt » est un autre choix possible). Puis entrez les commandes suivantes :

Listing 6 Décompression de l'archive

```
sudo cp hadoop-1.1.2.tar.gz /usr/local
cd /usr/local
sudo tar xzf hadoop-1.1.2.tar.gz
```

Changement des droits de propriété et du nom du répertoire

Ensuite il faut que l'utilisateur et le groupe d'Hadoop puissent exécuter les programmes. Pour cela on change le nom du répertoire, les droits sur celui-ci et le nom du propriétaire des fichiers Hadoop pour être l'utilisateur « hd_user » et le groupe « hadoop » :

Listing 7 Edition des droits

```
sudo mv /usr/local/hadoop-1.1.2 /usr/local/hadoop
sudo chmod -R 755 /usr/local/hadoop
sudo chown -R hd_user:hadoop /usr/local/hadoop
```

Configuration du fichier .bashrc et du PATH

Ensuite nous ajoutons dans le PATH les liens vers Hadoop afin que l'utilisateur hd_user puisse utiliser les commandes Hadoop sans taper le lien complet. Pour cela nous ajoutons les lignes suivantes à la fin du fichier « /home/hd_user/.bashrc », le fichier .bashrc est appelé à chaque login de l'utilisateur ce qui nous permet de modifier à chaque login le PATH de l'utilisateur hd_user :

Listing 8 Edition du .bashrc

```
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-6-sun
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
export PATH=$PATH:$HADOOP_HOME/bin
lzohead () { hadoop fs -cat \$1 | lzop -dc | head -1000 | less }
```

Configuration d'Hadoop

La configuration d'Hadoop se fait grâce à plusieurs fichiers qui permettent individuellement de modifier les paramètres d'un processus sous Hadoop, tous ces fichiers sont accessibles dans le répertoire $HADOOO_HOME/conf/ :

- Hadoop-env.sh : Variables d'environnement utilisées par Hadoop
- Core-site.xml : Configuration principale (comme les paramètres I/O pour le HDFS et MapReduce)
- Hdfs-site.xml : Configuration des démons Hadoop
- Mapred-site.xml : Configuration pour le démon MapReduce (JobTracker et les TaskTrackers)
- Masters : Liste des machines qui sont maîtres (NameNode et NameNode secondaire) sachant que dans notre cas ces deux composants se trouve sur la même machine.
- Slaves : Liste des machines qui sont les esclaves (DataNodes et TaskTrackers)

Nous allons maintenant nous intéresser au fichier « hadoop-env.sh ». Celui-ci permet de configurer tous les paramètres relatifs à Hadoop en lui-même. Dans ce fichier, on peut :

- Modifier l'allocation mémoire de chaque démon Hadoop
- Modifier l'emplacement Java (déterminé par le paramètre JAVA_HOME)
- Modifier l'emplacement des fichiers de logs (par le paramètre HADOOP_LOG_DIR)
- Personnaliser les paramètres SSH

Dans notre cas, nous avons juste besoin de modifier la variable JAVA_HOME pour qu'elle corresponde au chemin vers notre installation de Java 6. Nous modifions le fichier « /conf/hadoop-env.sh » et lui ajoutons la ligne suivante :

Listing 9 Définition de la variable JAVA HOME dans le fichier /conf/hadoop-env.sh

```
1 export JAVA\_HOME=/usr/lib/jvm/java-6-sun
```

Si jamais les interfaces réseau des serveurs exécutant Hadoop gère IPv6 il est préférable de le désactiver pour Hadoop. En effet il peut y avoir des problèmes avec IPV6. Pour cela il faut encore modifier le fichier /conf/hadoop-env.sh, recherchez la ligne suivante :

Listing 10 Désactivation de IPv6 pour Hadoop

```
1 # export HADOOP_OPTS=-server
2
3 # ajouter ceci
4 export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

Configuration du fichier core-site.xml

```
<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/tmp/hadoop</value>
        <description>A base for other temporary directories.</description>
    </property>

    <property>
        <name>fs.default.name</name>
        <value>hdfs://Machine1:54310</value>
        <description>The name of the default file system. A URI whose scheme
and authority determine the FileSystem implementation. The uri's scheme
determines the config property (fs.SCHEME.impl) naming the FileSystem
implementation class. The uri's authority is used to determine the host,
port, etc. for a filesystem.
        </description>
    </property>
</configuration>
```

FIGURE 2.1 – Fichier core-site.xml

Configuration du fichier HDFS-site.xml

```
<configuration>
    <property>
        <name> dfs.replication </name>
        <value>3</value>
        <description>Default block replication. The actual number of
replications can be specified when the file is created. The default is used
if replication is not specified in create time.
        </description>
    </property>
    <property>
        <name> dfs.name.dir </name>
        <value>/home/hd_user/hadoop/dfs/namenode </value>
    </property>
    <property>
        <name> dfs.data.dir </name>
        <value>/home/hd_user/hadoop/dfs/datanode </value>
    </property>
</configuration>
```

FIGURE 2.2 – Fichier HDFS-site.xml

Configuration du fichier mapred-site.xml

```
<configuration>
    <property>
        <name> mapred.job.tracker </name>
        <value> Machine1:54311</value>
        <description> The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map and reduce task.
        </description>
    </property>
    <property>
        <name> mapred.system.dir </name>
        <value>/home/hd_user/hadoop/mapred/system </value>
    </property>
</configuration>
```

FIGURE 2.3 – Fichier mapred-site.xml

Démarage d'Hadoop

Avant le lancement d'Hadoop le formatage de son système de fichier (HDFS) est nécessaire. Pour cela, il vous suffit d'entrer la commande suivante :

Listing 11 Formatage du système de fichier HDFS

```
1 hadoop namenode -format
```

```
hd_user@francois-HP-ProBook-4525s:/home/francois$ hadoop namenode -format
Warning: $HADOOP_HOME is deprecated.

13/04/21 22:47:06 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = francois-HP-ProBook-4525s/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.1.2
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.1 -r 1440782; compiled by 'hortonfo' on
************************************************************/
13/04/21 22:47:06 INFO util.GSet: VM type       = 64-bit
13/04/21 22:47:06 INFO util.GSet: 2% max memory = 19.33375 MB
13/04/21 22:47:06 INFO util.GSet: capacity      = 2^21 = 2097152 entries
13/04/21 22:47:06 INFO util.GSet: recommended=2097152, actual=2097152
13/04/21 22:47:07 INFO namenode.FSNamesystem: fsOwner=hd_user
13/04/21 22:47:07 INFO namenode.FSNamesystem: supergroup=supergroup
13/04/21 22:47:07 INFO namenode.FSNamesystem: isPermissionEnabled=true
13/04/21 22:47:07 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
13/04/21 22:47:07 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0
13/04/21 22:47:07 INFO namenode.NameNode: Caching file names occuring more than 10 times
13/04/21 22:47:07 ERROR namenode.NameNode: java.io.IOException: Cannot create directory /usr/local/tmp/hadoop/hadoop-hd_user/dfs
        at org.apache.hadoop.hdfs.server.common.Storage$StorageDirectory.clearDirectory(Storage.java:294)
        at org.apache.hadoop.hdfs.server.namenode.FSImage.format(FSImage.java:1326)
        at org.apache.hadoop.hdfs.server.namenode.FSImage.format(FSImage.java:1345)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.format(NameNode.java:1207)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.createNameNode(NameNode.java:1398)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.main(NameNode.java:1419)

13/04/21 22:47:07 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at francois-HP-ProBook-4525s/127.0.1.1
************************************************************/
```

FIGURE 2.4 – Résultat de la commande hadoop namenode –format

Si nous devons reformater le NameNode et supprimer les fichiers locaux HDFS via la commande suivante :

Listing 12 Formatage du système de fichier HDFS

```
1 rm /home/hd_user/hadoop/dfs/datanode /tmp/hadoop-hadoop -rf
```

Démarage du HDFS

Il nous suffit pour cela d'entrer la commande suivante :

Listing 13 Démarrage de HDFS

```
1 start-dfs.sh
```

Démarage de la partie MapReduce

Cela démarre également le JobTracker et les TaskTrackers. Pour cela, il suffit d'entrer la commande suivante :

Le JobTracker coordonne l'exécution des jobs sur l'ensemble du cluster. Il communique avec les TaskTrackers en leur attribuant des tâches d'exécution (map ou reduce).

Listing 14 Démarrage de MapReduce

```
1 start-mapred.sh
```

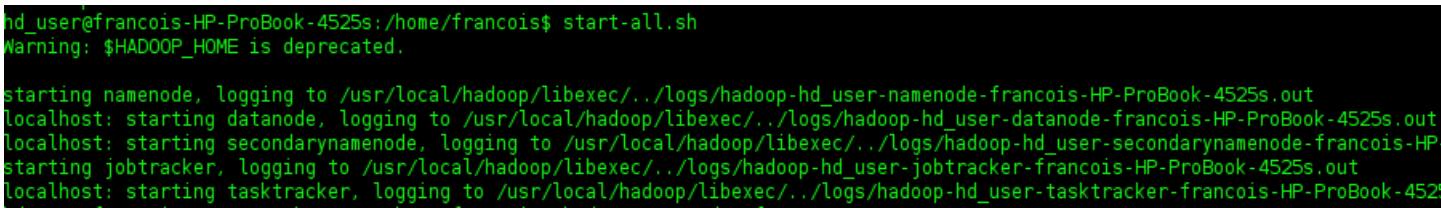

FIGURE 2.5 – Commande start-all.sh

Par ailleurs, il permet également d'avoir une vision globale sur la progression ou l'état du traitement distribué via une console d'administration web accessible par défaut sur le port 50030. C'est un démon qui cohabite avec le NameNode, c'est pourquoi il n'y a qu'une seule instance par cluster.
Les TaskTrackers exécutent les tâches (map ou reduce) au sein d'une nouvelle JVM instanciée par le TaskTracker. Par ailleurs, les jobs notifient périodiquement le JobTracker du niveau de progression d'une tâche ou bien le notifient en cas d'erreur afin que celui-ci puisse reprogrammer et assigner une nouvelle tâche. Un TaskTracker est un démon cohabitant avec un DataNode, c'est pourquoi il y a autant d'instances que de nœuds esclaves.

Vérification de l'état des Jobs

Il suffit pour cela d'entrer la commande suivante :

Listing 15 Liste des jobs MapReduce

```
1 hadoop job -list
```

La commande "jps", installée avec Java, nous permet de vérifier l'état de l'ensemble des composants Hadoop. Cependant si d'autres applications Java tournent sur la machine la commande jps va aussi afficher les informations sur ces applications.

Voici les résultats de ces commandes :

```
hd_user@francois-HP-ProBook-4525s:/home/francois$ hadoop job -list
Warning: $HADOOP_HOME is deprecated.

0 jobs currently running
JobId   State   StartTime       UserName        Priority        SchedulingInfo
hd_user@francois-HP-ProBook-4525s:/home/francois$ jps
5437 Jps
3308 SecondaryNameNode
3392 JobTracker
```

FIGURE 2.6 – Commande hadoop job -list et jps

Visualisation des interfaces utilisateurs

Hadoop propose un ensemble d'interfaces web permettant de monitorer les divers applications associés à Hadoop. Pour y accéder il faut entrer les URL ci-dessous dans un navigateur web classique. On peut par exemple voir comment est configuré HDFS avec les différents noeuds associés mais aussi consulter des logs :

- http://localhost:50030/ pour le JobTraker
- http://localhost:50060/ Pour les tasktracker(s)
- http://localhost:50070/ Pour le(s) NameNode(s) d'HDFS

L'interface web du JobTraker permet de récupérer quelques métriques tels que les jobs et les tâches en cours d'exécution sur le cluster ainsi qu'un historique sur les jobs terminés (y compris ceux qui ont échoué)

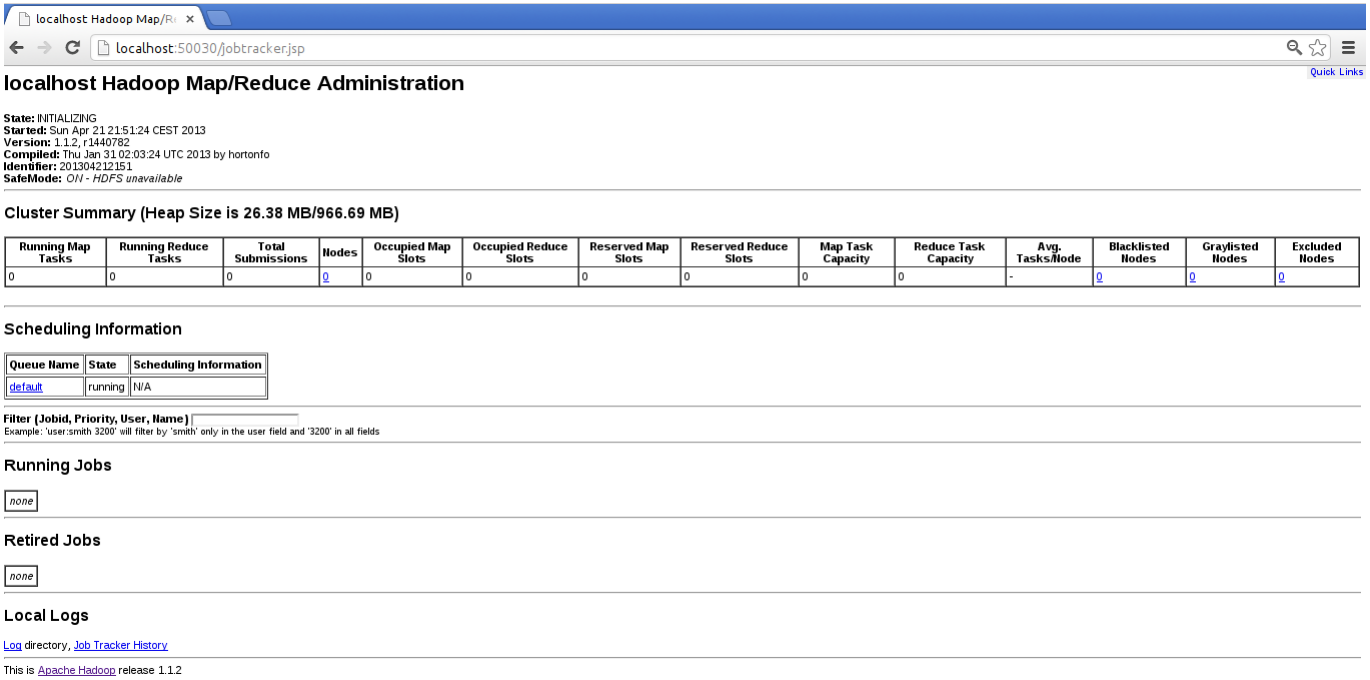

FIGURE 2.7 – Interface web du JobTraker

Il n'y a pas de job en cours dans la capture d'écran ci-dessus, mais si il y en avait nous serions en mesure de visualiser l'état actuel de ces job (y compris les compteurs pour les dossiers d'entrée/sortie de

lecture, la localité d'exécution d'une tâche, etc.). Au bas de cette page, il y a un lien vers l'historique du JobTracker qui nous permet de visualiser les statistiques sur les travaux terminés.

Exécution d'un Job MapReduce

Pour exécuter notre propre Job, il faut compresser le programme en « .jar ». Et enfin, exécuter la commande suivante :

Listing 16 Exécution d'un jar contenant un Job MapReduce

```
1 hadoop jar leJar.jar nomClassJava repertoire repertoire-output
```

Un peu de précision sur la commande ci-dessus :
- "leJar.jar" est le Job que l'on veut exécuter
- "nomClassJava" est le nom de la classe Java
- "repertoire repertoire-output" est le nom du répertoire où l'on veut stocker le résultat du traitement.

Arrêt de Hadoop

Listing 17 Arret des processus Hadoop

```
1 stop-dfs.sh
2 stop-mapred.sh
```

Déroulement du projet

Pour la réalisation de ce projet nous avons suivi au maximum notre planning prévisionnel du cahier des charges ; celui-ci a été respecté dans son ensemble.

Nous avons utilisé une méthodologie SCRUM pour le projet. C'est pourquoi, dans un premier temps, nous avons réalisé un découpage en tâche qui a été validé par notre encadrant. Voici le découpage qui a été réalisé :

Nom	Installation Hadoop
ID	THAB
Description	On installe Hadoop et on s'assure de son bon fonctionnement

Nom	Configuration Hadoop
ID	CHAD
Description	Configuration Hadoop en mode local

Nom	Comprendre le fonctionnement MapReduce
ID	MAP
Description	Comprendre le modèle de programmation parallèle qui facilite le développement d'applications distribuées

Nom	Comprendre un exemple simple
ID	EXE
Description	Comprendre un exemple simple le wordcount

Nom	Mise en place de la fonction MapReduce
ID	FMAP
Description	Codage de la fonction MapReduce (fonction cosinus qui va trier nos documents)

Nom	Implémenter un comparateur de documents textuels
ID	ICOM
Description	Implémentation d'un comparateur de documents textuels qui s'appuiera sur la mesure de similarité de la fonction cosinus

Nom	Test et modification en fonction des résultats de la fonction MapReduce
ID	PRES
Description	Présentation au client des différents résultats obtenus

Nom	Rédaction du rapport et préparation de la soutenance
ID	CRAP
Description	écriture du rapport et préparation de la soutenance

Nom	Installation du mode pseudo-distribué
ID	PDIS
Description	Installation du mode pseudo-distribué afin de simuler le fonctionnement en mode cluster où les daemons Hadoop sont exécutés

Au départ nous nous sommes donc concentré sur l'installation et la configuration de Hadoop. Ensuite, nous nous sommes documenté afin de comprendre le fonctionnement de Hadoop et plus particulièrement la fonction MapReduce de celui-ci. Pour cela, nous avons choisi un exemple simple, l'exemple du wordcount qui sera présenté en détail dans la suite de ce présent document.

Ensuite, après avoir bien assimilé le fonctionnement MapReduce nous nous sommes concentré sur le comparateur de documents textuels que nous devions mettre en place. Pour la réalisation de celui-ci nous nous sommes donc concerté afin de nous répartir le travail sur les différentes parties du comparateur. Nous avons uni nos compétences afin que chaque partie du travail soit comprises par les deux binômes. Dès que l'un de nous bloquait sur quelque chose, il n'hésitait pas à demander conseil à l'autre binôme. Nous avons également eu l'occasion de voir régulièrement notre encadrant afin de savoir si nous partions sur le bon chemin.

De plus, nous avons perdu du temps car nous avons utilisé une librairie java de Hadoop ancienne qui était un peu limitée par rapport à la dernière version de Hadoop que nous utilisions (ce problème vous sera expliqué plus en détail dans la suite de ce rapport). Après avoir rectifié ce problème, nous avons pu tester notre programme via les documents textuels récupérés lors des TP de traitements des données multimédia. Et enfin, faire valider la tâche par notre client lors de la courte présentation que nous lui avons faîte.

Et pour finir, nous avons cherché à tester notre programme en mode pseudo-distribué afin de simuler le fonctionnement en mode cluster.

Résultat du projet

4.1 Implémentation de MapReduce

MapReduce est un modèle de programmation parallèle introduit par Google pour traiter de gros volumes de données. Le modèle est relativement simple. Dans Hadoop, il est implémenté sous forme d'un framework performant pour effectuer du traitement en batch sur des données pouvant être découpées en blocs indépendants. Par conséquent, MapReduce ne convient pas à tous les usages.

MapReduce a été conçu pour remplir trois objectifs :
- fournir une API simple pour que les développeurs fassent du code distribué
- paralléliser en petits traitements unitaires de très gros volumes de données
- déplacer le code vers la donnée plutôt que le contraire

Ce dernier point est fondamental. C'est ce qui différencie MapReduce des autres frameworks de calcul distribué. En effet le Job est compilé en JAR et est déployé sur tous les datanode du cluster afin d'exécuter le code au plus prêt des données. Celui-ci est un Jar standard contenant un Main et 2 classes implémentant chacune une interface : Mapper et Reducer. Le tout est envoyé comme job au cluster Hadoop via le JobTracker.

Un traitement MapReduce est découpé en trois phases qui sont les suivantes :
- Le map
- Le shuffle & merge
- Le reduce

Toutes les données traitées par toutes les phases sont du type clé valeurs : (input) <k1, v1> → map → <k2, v2> → shuffle/merge → <k2, v2> → reduce → <k3, v3> (output). Schématiquement le traitement peut être illustré de la sorte :

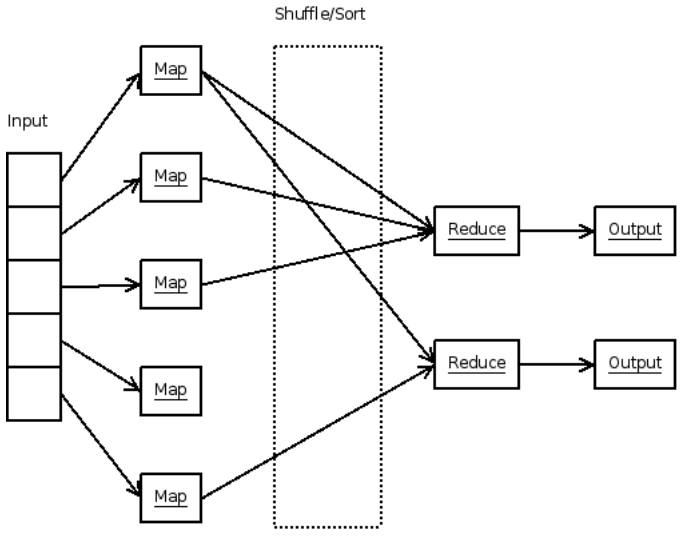

FIGURE 4.1 – schéma mapreduce

La phase de Map

Comme on peut le voir sur le schéma pour chaque entrée une Map est créée. Celle-ci est une étape de chargement et de transformation des données sous la forme de paires clé/valeur.

La phase de shuffle & merge

Cette deuxième phase intervient une fois que toutes les map sont terminées. Elle va trier au moyen d'un quick-sort et regrouper les valeurs du résultat intermédiaire par clé. Et chaque clé est envoyée avec sa liste de valeur à une JVM de reduce.

La phase de Reduce

Et enfin la dernière phase (Reduce) est une étape de fusion des enregistrements par clé pour former le résultat final. Ainsi, et c'est un principe qui est au cœur de Hadoop, les données à traiter en entrée sont découpées en "petites" unités, chacune étant traitée en parallèle par une fonction Map. Le résultat des traitements unitaires est trié par clé pour former des unités de données passées à une fonction Reduce. C'est parce que les programmes MapReduce sont intrinsèquement prévus pour être exécutés en parallèle qu'il est possible de répartir le traitement.

4.1.1 Un exemple simple

Afin de mieux comprendre le fonctionnement MapReduce, nous allons prendre un exemple simple. Pour cela, nous avons choisi de prendre l'exemple du wordCount. Le but de cet exemple est de prendre plusieurs fichiers d'entrée et de compter pour chaque mot le nombre de fois qu'il est utilisé sur l'ensemble des fichiers.

Notre exemple se décompose en trois classes. Tout d'abord la classe WordCountMapper qui implémente l'interface Mapper fournie par l'API Hadoop. La classe WordCountMapper possède une méthode map qui prend en entrée une ligne d'un fichier. Pour chaque ligne nous allons découpé chaque mot puis associé à chaque mot le chiffre 1. Nous aurons donc pour chaque map des couples <mot, 1>, dans la phase de merge/sort toutes les clés identiques, donc dans notre exemple les mots, vont être regroupés. Ces regroupements vont être passés au reducer, chaque reducer aura en paramètre une clé et un ensemble de valeurs, nous n'auront plus qu'à additionner nos 1 pour connaître pour chaque clé/mot le nombre d'occurrence. C'est aussi la phase de reduce qui va permettre la création du fichier de sortie, celui-ci sera aussi un ensemble de couple <clé, valeur> stocké dans un fichier.

Et enfin, le client est une simple classe Java avec une méthode main(), cette méthode va initialiser le job avec comme paramètres : le fichier de données à traiter, le répertoire de stockage des fichiers résultats, les classes Mapper et Reducer à utiliser. Plein d'autres choses peuvent être paramétré, on peut choisir de définir nous même les shuffles et merges, on peut définir une étape supplémentaire entre le Mapper et le Reducer qui s'appelle Combiner. Cette étape est une phase de reduce appliquée sur chacune des Map, cela permet par exemple dans notre exemple de word count de faire une première addition des occurrences d'un mot sur chaque ligne avant de faire l'addition pour l'ensemble des fichiers.

Le lancement du job se fait comme décrit dans la section .Ci-après le code pour réaliser un word count :

Listing 18 WordCount Mapper

```
class WordCountMapper extends MapReduceBase implements Mapper    {
  //Preallocation pour reutiliser d'un appel a l'autre
  private final static IntWritable one = new IntWritable(1)
  private Text word = new Text()
  /**
   *  Map
   *  @param key l'identifiant de la ligne
   *  @param value le contenu de la ligne
   */
  public void map(LongWritable key, Text value, Context context)
    throws IOException {
    String lLine = value.toString()
    /* decoupage de la ligne par mots */
    StringTokenizer tokenizer = new StringTokenizer(lLine)
    while(tokenizer.hasMoreTokens())  {
      word.set(tokenizer.nextToken())
      /* creation du couple <mots, 1> */
      context.write(word, one)
    }
  }
}
```

Listing 19 WordCount Reducer

```
class WordCountReducer extends MapReduceBase implements Reducer  {

  /**
   *  Reduce
   *  @param key la cle
   *  @param values l'ensemble des valeurs associe pendant la phase de Map
   */
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException {
    int sum = 0
    // iteration sur les valeurs pour additionner les occurrences du mot
    for(IntWritable val : values)  {
      sum = sum + val.get()
    }
    // ecriture du couple <mot, nombre d'occurrence>
    context.write(key, new IntWritable(sum))
  }
}
```

Listing 20 WordCount Main

```
class WordCount  {
  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class)
    conf.setJobName("Compteur de mots")
    //On positionne les types du resultat
    conf.setOutputKeyClass(Text.class)
    conf.setOutputValueClass(IntWritable.class)
    //On indique quel est la classe du mapper et celle du reducer
    conf.setMapperClass(WordCountMapper.class)
    conf.setReducerClass(WordCountReducer.class)
    //On indique quel est le type de formatage des fichiers d'entree et de sortie
    conf.setInputFormat(TextInputFormat.class)
    conf.setOutputFormat(TextOutputFormat.class)
    //On dit au framework de lire le repertoire args[0] en entree
    //Et d'ecrire dans args[1] en sortie
    FileInputFormat.setInputPaths(conf, new Path(args[0]))
    FileOutputFormat.setOutputPath(conf, new Path(args[1]))
    JobClient.runJob(conf)
  }
}
```

4.2 Réalisation de notre projet en lui-même

Le but principal de notre projet mise à part la compréhension d'un job MapReduce était de mettre en place un système comparaison de document textuel avec une méthode cosinus. Ce système nous permet de créer un pseudo moteur de recherche en indexant tous les documents et en les comparant à une question nous pouvons ressortir un niveau de similarité. Ce problème est décomposé en 2 sous-problèmes. Premièrement la création d'une indexation de tous les documents, puis la mise en place du moteur de requêtage.

4.2.1 index inversé

Pour le premier sous problème nous avons utilisé la méthode de l'index inversé. Cette méthode permet de faire l'association entre du contenu et un ensemble de données. Voici l'exemple donné par Wikipédia : Soient les documents suivant D1 = "L'art de plaire est l'art de tromper.", D2 = "Nous avons l'art, afin de ne pas mourir de la verité.", un index inversé simple aurait cette forme :

Listing 21 Exemple d'index inversé

```
"art"        {D1, D2}
"plaire"     {D1}
"est"        {D1}
"tromper"    {D1}
"nous"       {D2}
"avons"      {D2}
"afin"       {D2}
"pas"        {D2}
"mourir"     {D2}
"verite"     {D2}
```

Cependant l'index inversé tel qu'il est décrit par wikipédia ne nous permet pas de faire des calculs de TFIDF. Par la suite nous avons donc utilisé ce principe en ajoutant des informations dans le fichier tel que le nombre de document utilisant le signe, pour la partie DF, et les nombre d'occurrence du signe dans un document pour la partie TF. Pour créer cet index nous avons fait un job mapreduce qui prend en entrée la liste de tous les fichiers et créer cet index inversé.

La partie Map est exécutée sur toutes les lignes de chaque fichier, on extrait les signes et on crée les couples <signe, nomFichier>. Dans la phase de reduce on récupère donc pour chaque signe tous les documents qui l'utilise, nous utilisons un objet HashMap afin de stocker pour chaque document le nombre de fois qu'il apparait dans cette liste. Ensuite nous n'avons plus qu'a écrire dans le fichier de sortie le couple <sign, df fichier=>tf>.

Listing 22 Extrait de l'index inverse genere

```
a  2  poisson_4.txt=>11 chat_2.txt=>9
affirme  1  poisson_4.txt=>1
age  1  poisson_4.txt=>1
algerie  1  poisson_4.txt=>1
algeriennes  1  poisson_4.txt=>1
algeriens  1  poisson_4.txt=>1
alimenter  1  chat_2.txt=>1
animal  1  poisson_4.txt=>1
```

4.2.2 Query Job

La deuxième partie du problème est de calculer des distances entre des documents représentés sous forme vectoriel et une question. Pour cela nous utilisons une distance cosinus. Il s'agit de la mesure de similarité la plus utilisée en traitement de texte. Celle-ci calcule l'angle entre 2 vecteurs, cet angle étant compris entre 0 et π, une distance de π indiquera des vecteurs opposés, $\frac{\pi}{2}$ des vecteurs orthogonaux et 0 des vecteurs colinéaires. Soit A et B deux vecteurs l'angle θ est obtenu avec la formule suivante :

$$\theta = \arccos \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{4.1}$$

Pour représenter un document textuel sous forme de vecteur on utilise l'approche sac de mots. Ces vecteurs ne tiennent pas compte de la structure des documents et visent juste à représenter les documents comme un histogramme de l'importance de chacun des mots du vocabulaire dans un document. Étant donné que l'on considère tous les mots d'un vocabulaire, nos vecteurs font parties d'un espace moyen de dimension R^{100000} pour des documents monolingues, nous n'allons donc pas créer les vecteurs en mémoire.

Pour le calcul des vecteurs nous utilisons directement l'index inversé et une question passée en paramètre du Jar. Pour que la question soit passée à chacune des Map et des reducers on utilise l'objet context qui est passé à tous les traitements. Cet objet stocke diverses informations telles que le nom du job, nous avons donc choisi d'utiliser le nom du job pour stocker la question déjà sous forme vectoriel et ainsi y accéder sur toutes les Map et les Reduce. Nous n'avons pas trouvé d'autres moyens de passer des paramètres entre les Map, à moins de réécrire le fichier d'entrée pour qu'il contienne sur toutes les lignes le paramètre en question.

Nous mappons sur toutes les lignes de l'index inversé, sur chaque ligne nous créons un couple <fichier, word-DF-TF>. C'est dans la phase de reduce que le calcul de la distance cosinus est fait. Dans cette phase nous récupérons le fichier avec l'ensemble des signes utilisés par celui-ci. En itérant sur les signes on itère sur les composantes non-nulles de notre vecteur et on peut calculer la norme du vecteur du fichier et le produit scalaire avec la question. Le fichier de sortie est l'association du nom du fichier avec sa valeur de similarité avec la question.

Listing 23 Exemple d'un fichier de sortie

```
1 chat_2.txt  0.14793410181646144
2 poisson_4.txt  0.20172297194417788
```

Le code complet et disponible en annexe.

Problèmes rencontrés

Lors de ce projet nous avons rencontré différents problèmes ce qui nous paraît tout à fait normal.

Tout d'abord, nous avons essayé dans un premier temps de travailler sur un serveur distant hébergé chez un particulier. Cependant nous avons rencontré des problèmes avec les ports utilisés par Hadoop, il fallait débloquer plusieurs ports sur la box pour accéder à toutes les fonctionnalités d'hadoop et cela était difficile à gérer à distance. C'est pourquoi, nous avons essayé d'installer Hadoop sous Windows, mais une fois encore nous avons rencontré des problèmes. Nous avons utilisé cygwin qui permet d'émuler l'invite de commandes de Linux sous windows mais impossible de faire fonctionner Hadoop, nous avons rencontré des problèmes de configuration car celle-ci n'était pas compatible de base sous windows. Donc la solution a été de l'installer sous Linux, pour cela nous avons choisit une distribution UBUNTU 12.04.

Pour finir, comme nous avions réussit l'ensemble des tâches qui nous était demandé de réaliser pour ce projet d'option, nous avons essayé de réaliser la partie "optionnel" concernant la mise en place du mode pseudo-distribué. En effet, l'exécution du programme en mode distribué posait un problème avec le jar. Ce problème n'a pas été résolu pour le moment. Toutefois, nous avons réussi à exécuter un wordcount simple en mode pseudo distribué. Nous pensons que le problème est dû à la manière dont nous générons le JAR. Pour réaliser le projet nous avons créer un projet maven avec comme dépendance les jar d'hadoop, cela nous permet de nous partager les sources sans problème de classpath ou de lib inexistante étant donnée que maven gère tout cela pour nous. Cependant nous avons utilisé des dépendances qui ne sont pas propre à Hadoop qui nous permettent de gérer un fichier de configuration passé en paramètre du JAR permettant à la personne utilisant notre système de configurer la manière dont l'extraction des signes est faite : méthode simple des sacs de mots ou sac de n-grammes. Ce ficher de configuration permet aussi d'appliquer des filtres sur les signes comme supprimer les stop-words, appliquer un stemming, etc. Lorsque l'on génère le Jar via l'utilitaire maven mvn et que l'on exécute le jar dans Hadoop celui-ci ne trouve pas les dépendances supplémentaires et si l'on compile dans le jar toutes les dépendances, cela ne marche pas non plus.

Dans l'ensemble, les problèmes rencontrés ne nous ont pas tant retardé et nous avons pu finir dans les délais.

Remerciements

Nous tenons à remercier M. Romain Raveaux pour sa disponibilité tout au long de ce projet, que ce soit pour des problèmes que nous avons rencontrés ou encore pour faire le point sur l'avancement au fil du temps.

Conclusion

Malgré quelques problèmes , nous avons réussi à réaliser le projet qui nous a été confié dans les délais.

Ce projet nous a permis d'apprendre beaucoup de notions en ce qui concerne le Big Data. La réalisation de celui-ci nous a permis de découvrir et de nous familiariser avec les moteurs d'indexation et de mieux comprendre leur fonctionnement. Ce projet nous a également permis de mettre en place les notions vues au premier semestre avec les différents intervenants ou la conférence sur le Big Data. Cela nous a aussi permis de programmer et d'utiliser les divers algorithmes et notions vues en cours de traitement des données dans un cas concret.

De plus, ce projet est très intéressant puisque cela nous forme bien à notre futur métier d'ingénieur, en effet nous avons utilisé un framework que nous ne connaissions pas.

Finalement, nous ne pouvons tirer que des conclusions positives sur l'expérience que nous a apporté ce projet tant sur les relations humaines que sur l'apprentissage du JAVA et du framework Hadoop.

Bibliographie

– *Les cours de traitements des données de M.* Raveaux *Romain.*

– *La documentation mise en ligne par apache sur leur site.*

Hadoop : Mise en application du concept MapReduce

Département Informatique
5e année
2012 - 2013

Rapport Projet Option Web & Multimédia

Résumé : Projet d'option Web & Multimédia qui consiste à prendre en main l'outil Hadoop mis en place par Apache. Afin de mettre en place un modèle MapReduce qui est un modèle de programmation parallèle facilitant le développement d'applications distribuées. Nous devions implémenter un comparateur de documents textuels qui s'appuiera sur la mesure de similarité Cosinus.

Mots clefs : Hadoop, projet option Web & multimédia, MapReduce, programmation parallèle

Abstract: Project Web & Multimedia option of taking in hand the Hadoop tool developed by Apache. To develop a MapReduce's model who is a parallel programming model for easy development of distributed applications. We had to implement a textual comparison that will build on the Cosine similarity measure.

Keywords: Hadoop, Web & multimedia project option, MapReduce parallel programming

---

Encadrant
Romain RAVEAUX
romain.raveaux@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Jérôme HEISSLER
jerome.heissler@etu.univ-tours.fr
François SENIS
francois.senis@etu.univ-tours.fr

DI5 2012 - 2013