REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE BADJI MOKHTAR ANNABA
FACULTE DES SCIENCES DE L'INGENIORAT
DEPARTEMENT INFORMATIQUE

HADOOP MAPREDUCE

Master : Gestion et Analyse des Données Massives (GADM)

2^{eme}année

Dr. Klai Sihem

Avant propos...

Le Big Data est une science récente qui a surgie avec l'évolution et la variation des données et d'applications mises et échangées en ligne. Cette science consiste à prendre en charge d'une manière efficace un volume important des données hétérogènes en intégrant des techniques et outils nouveaux, vu que la technologie disponible ne répond plus aux besoins.
Ce polycopié est un support pédagogique qui permet d'initier l'étudiant au domaine des Big Data. Ce cours composé de plusieurs chapitres permet aux étudiants de comprendre la problématique et la motivation du domaine, et de maîtriser l'outil Hadoop avec le modèle MapReduce associé à ce domaine.

Chaque chapitre est élaboré pour répondre à un but pédagogique bien précis, se matérialisant par des explications, définitions accompagnées d'exemples et des illustrations par des figures suivies par des exercices, des solutions envisageables ou des fiches de travaux pratiques bien guidés.

1. Le chapitre I met l'étudiant dans le contexte du Big Data, consiste à lui donner des connaissances générales sur le domaine ;
2. Le chapitre II est consacré à l'étude de Hadoop, le framework qui permet le développement d'applications traitant les données massives. Ce chapitre donne les notions les plus générales avec la procédure d'installation du logiciel ;
3. Le chapitre III détaille la partie qui s'occupe du stockage des données "HDFS", avec la possibilité de la manipulation de ces données selon deux manières différentes à savoir : les commandes et l'API JAVA ;

4. Le chapitre IV étudie en détail la partie traitement des données massives "MapReduce", le modèle qui permet de traiter des blocs de données séparément et parallèlement dans des machines connectées. La modélisation selon le paradigme MapReduce est une étape importante avant le développement des programmes ;
5. Le chapitre V détaille l'implémentation des programmes MapReduce dans Hadoop. Dans le cadre de ce chapitre, nous étudions l'implémentation des programmes en utilisant le langage Java. D'autres langages peuvent être utilisés pour écrire des programmes mapreduce, mais cette partie n'est pas traiteé dans ce cours.

L'élaboration de ce polycopié a été inspirée de plusieurs documents, j'ai pris le soin de les citer dans la partie bibliographie, d'autres documents aussi ont été cité afin d'apporter aux lecteurs plus de commandes et plus de détails sur les parties traitées dans ce cours.

Terminologie. . .

JVM : Java virtuelle machine

HDFS : Hadoop Distributed File System

YARN : Yet Another Ressource Negociator

AM : Application Master

API java : Application Programming Interfaces java

Table des matières

Liste des figures

1 MapReduce : Présentation et Modélisation

Objectif Pédagogique

A la fin de ce chapitre, l'étudiant sera capable de modéliser des problèmes selon le paradigme Mapreduce.

Pré-requis

L'étudiant doit :

— Avoir des connaissances sur le paradigme de la programmation fonctionnelle et l'algorithmique en général ;

— Maîtriser les connaissances acquises lors des chapitres précédents tels que Hadoop et HDFS.

Dans ce chapitre, nous allons étudier MapReduce, un paradigme qui permet de modéliser des problèmes selon les fonctions Map() et Reduce() afin d'automatiser le traitement parallèle et distribué sur des données massives.

1.1 Historique

Ce modèle a été proposé dans les années 2000, par deux ingénieurs de chez Google, qui ont observé qu'un grand nombre des traitements massivement parallèles, mis en place pour les besoins de leur moteur de recherche, suivaient une stratégie de parallélisation identique. De ces observations est né le modèle de programmation MapReduce, décrit pour la première fois en 2004 dans un article de recherche (voir Article 15). Son principe général est que toute parallélisation de traitement sur des données massives peut s'effectuer uniquement à l'aide de deux types d'opérations : une opération map et une opération reduce 06.

1.2 Présentation de MapReduce

Le paradigme (modèle) MapReduce est principalement utilisé pour le traitement distribué sur de gros volumes de données aux seins d'un cluster de nœuds. Il tolère la scalabilité en ajoutant des nœuds dans le cluster et tolère aussi les pannes avec son système de réplication.

Considérons qu'on a à traiter un fichier de données de très grande taille dont le traitement classique prend énormément de temps. Pour résoudre ce problème, on utilise MapReduce qui permet de faire le traitement sur des fragments du fichier en entrée, séparément et parallèlement.

Comme son nom l'indique, MapReduce est la composition des fonctions Map() et Reduce(). Utiliser MapReduce revient à modéliser le traitement se-

lon le paradigme MapReduce, ceci revient à le modéliser selon ces deux fonctions.

1. La fonction MAP() est appliquée sur les fragments (une Map par fragment) et fournit des résultats intermédiaires ;
2. La fonction Reduce() traite les sorties des Map(), agrège les résultats intermédiaires et fournit un résultat final.

Pour comprendre le modèle MapReduce, il faut d'abord comprendre les concepts suivants : Le format des données, un format particulier représenté par des paires (clé, valeur) et le principe des fonctions Map() et Reduce().
Commençons par détailler le format des données échangées entre Map et Reduce

1.2.1 Format des données : paires (clé,valeur)

Les données échangées entre Map et Reduce, et dans la totalité du job sont des paires (clé, valeur) :

— une clé : de type quelconque : entier, texte,...

— une valeur : de type quelconque.

A titre d'exemple, on peut considérer un fichier texte comme un ensemble de (n° de ligne, ligne), selon le format paire(clé, valeur), où le n° de ligne est la clé et la ligne représente la valeur correspondante à cette clé.
Les deux fonctions Map() et Reduce() reçoivent des paires (clé, valeur) et émettent d'autres paires dans le même format (clé, valeur), selon les besoins de l'algorithme concernant le problème à traiter.

1.2.2 Principe de MapReduce et inspiration fonctionnelle

MapReduce s'inspire largement du paradigme de la programmation fonctionnelle qui donne un rôle central à la programmation et plus particulièrement aux opérateurs de liste mapreduce. En programmation fonctionnelle 16) :

1. map consiste à appliquer une même fonction à tous les éléments de la liste :
   — $map(f)[x0,...,xn] = [f(x0),...,f(xn)]$,

Exemple 1.1 $map(*4)[2,3,6] = [8,12,24]$

2. reduce applique une fonction récursivement à une liste et retourne un seul résultat :
   — reduce(f)[xo,...,xn]=f(xo,f(x1,f(x2,...))),

Exemple 1.2 *reduce(+)[2,3,6]=(2+(3+6))=11*

map et reduce sont des opérateurs génériques et leur combinaison permet de modéliser énormément de problèmes 06). Nous allons maintenant expliquer le modèle de programmation MapReduce.

1.3 Modélisation des problèmes selon le paradigme MapReduce

1.3.1 Exemple introductif

Le but à travers cet exemple est d'introduire les étapes de la modélisation en les appliquant sur un jeu de données. L'exemple porte sur une liste de joueurs définis par leurs scores, on désire calculer le score total pour chaque joueur.

Exemple 1.3 *Soient les* 4 *tuples suivants, (voir table 1.1) :*

ID-joueur	Score
joueur1	2
joueur2	4
joueur1	3
joueur2	2

Table 1.1 – *Exemple introductif*

Nous allons d'abord découper le fichier de données en 4 fragments, chaque fragment contient un tuple (une ligne). Et chaque tuple est décrit par :

- Le premier tuple noté $t1$ contient la valeur(ID-joueur= $joueur1$, $score = 2$);
- Le deuxième tuple noté $t2$ contient la valeur(ID-joueur= $joueur2$, $score = 4$);
- Le troisième tuple noté $t3$ contient la valeur(ID-joueur= $joueur1$, $score = 3$);
- Le quatrième tuple noté $t4$ contient la valeur(ID-joueur= $joueur2$, $score = 2$);

L'étape map :

Pour rendre le problème sous la forme de la fonction map(), nous allons nous poser la question suivante : Quel est le traitement qui se répète pour chaque tuple afin de calculer le score total de chaque joueur ?

Le traitement qui se répète pour chaque tuple est : la lecture du N du joueur et de son score, alors, si on veut formuler notre problème selon le principe de la fonction map(), on aura :
$map(f)[t1, t2, t3, t4] = [f(t1), f(t2), f(t3), f(t4)]$, est la fonction f, c'est le traitement qui se répète c.a.d (la lecture du N du joueur et de son score), alors :
$map(lire(ID{-}joueur, score))[t1, t2, t3, t4] =$
$map(lire(ID{-}joueur, score)[t1]), map(lire(ID{-}joueur, score)[t2]),$
$map(lire(ID{-}joueur, score)[t3]), map(lire(ID{-}joueur, score)[t4]) =$
$[lire(ID{-}joueur = joueur1, score = 2),$
$lire(ID{-}joueur = joueur2, score = 4),$
$lire(ID{-}joueur = joueur1, score = 3),$

$lire(ID-joueur = joueur2, score = 2)]$.

Les quatre map (map(t1), map(t2), map(t3), map(t4)) peuvent se faire simultanément, par exemple sur 4 machines différentes, à condition que chaque machine contient les données nécessaires pour le traitement.

L'étape reduce :

Calculer le score total revient à faire une agrégation des résultats de la fonction map dans la fonction reduce, cette agrégation consiste à faire la somme (c'est la fonction reduce f) des scores pour le même N°de joueur, cela consiste à :

- récupérer les résultats des map() pour ID-joueur= $joueur1$ et appliquer la fonction reduce() :
  reduce(f)[score-joueur1, score-joueur1]= reduce(+)[2,3]=(2+3)=5 ;
- récupérer les résultats des map() pour ID-joueur= $joueur2$ et appliquer la fonction reduce() :
  reduce(f)[score-joueur2,score-joueur2]= reduce(+)[4,2]=(4+2)=6.

Important : on remarque qu'une étape intermédiaire entre map et reduce est nécessaire, c'est l'étape qui fait le regroupement des scores par joueur.
Donc, on peut déduire que :

1. L'entrée de map est une paire (clé=n° tuple, valeur= tuple) ;
2. La sortie du map est une paire (clé= ID-joueur, valeur= score) ;
3. L'entrée de reduce est une paire (clé= ID-joueur, valeur= liste des scores correspondants à ce joueur) ;
4. La sortie de reduce est une paire (clé=ID-joueur, valeur= somme des scores de la liste).

On peut généraliser ces étapes afin de donner une démarche à suivre pour modéliser un problème selon le paradigme Mapreduce.

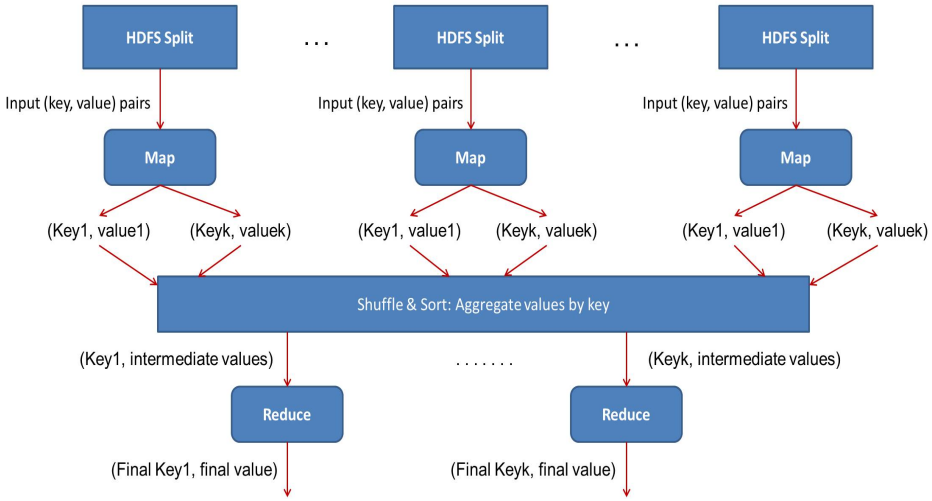

FIGURE 1.1 – *Démarche de modélisation selon Mapreduce* 17

1.3.2 Démarche de modélisation selon le paradigme Mapreduce

Les étapes à suivre pour modéliser un problème selon le paradigme MapReduce sont, (figure 1.1) :

1. Découper (split) le fichier d'entrée en fragments, chaque fragment est caractérisé par (son numéro et son contenu) , selon le format (clé, valeur). Exemple : numéro=$t1$, valeur=(ID-joueur= $joueur1$, $score = 2$) ;
2. Déterminer le traitement qui se répète pour tous les fragments, ce traitement fera l'objet de la fonction map, exemple "la lecture du N du joueur et de son score" ;
3. On applique un map pour chaque fragment, la paire (numéro, valeur) du fragment est l'entrée du map. Les map sont traités séparément et parallèlement ;
4. Au niveau du map, les données d'entrée du fragment sont transformées en une série de paires (clé,valeur) de telle sorte que ces paires (clé,valeur) aient un sens par rapport au problème à résoudre, exemple clé=ID-joueur=joueur1, valeur=$score = 2$;
5. La fonction Map s'écrit de la manière suivante :
   Map(cléE,valeurE) $\rightarrow$ Liste(cléI,valeurI) avec cléE et valeurE représentent la clé et la valeur en entrée du Map respectivement, et cléI, va-

leurI représentent la clé et la valeur en sotie du Map respectivement et qui sont aussi des résultats intermédiaires ;

6. Avant d'être envoyés à l'étape Reduce, une étape de tri et organisation (SHUFFLE and SORT) permet de regrouper les résultats intermédiaires par clé. Exemple : clé=ID-joueur=joueur1, valeur=[$score = 2, score = 3$] ;
7. L'étape Reduce accepte en entrée la liste des valeurs d'une même clé, exemple reduce(clé=joueur1, valeur=[2,3]) ;
8. L'étape Reduce agrège les résultats intermédiaires afin de donner le résultat unique à une clé intermédiaire. Exemple : reduce(joueur1, [2,3]) = (joueur1, 5) ;
9. La fonction Reduce s'écrit de la manière suivante :
   Reduce(cléI, List(valeurI)) $\rightarrow$ résultat .

1.3.3 Exemple de WordCount

"WordCount" est un exemple typique de MapRedue et du calcul distribué. Prenons en entrée un fichier texte, l'objectif de WordCount est de calculer le nombre d'occurrences de chaque mot dans le fichier. Si le texte est très grand à l'image de la collection Wikipedia qui contient environ 27 milliards de mots (source :Wikipedia), la solution séquentielle ne suffira pas et il est nécessaire de réaliser ce comptage de manière distribuée et parallèle. C'est là qu'intervient MapReduce, figure 1.2 :

1. Nous allons donc supposer que nos données d'entrée ont été découpées en différents fragments soit un fragment par ligne. Nous pouvons représenter facilement ces fragments sous la forme de paires (clé, valeur), en prenant comme clé le numéro de ligne et comme valeur la chaîne de caractères correspondant à la ligne à savoir :
   — ***ligne1= "The cat sat on the mat"***
   — ***ligne2= "The aardvark sat on the sofa"***

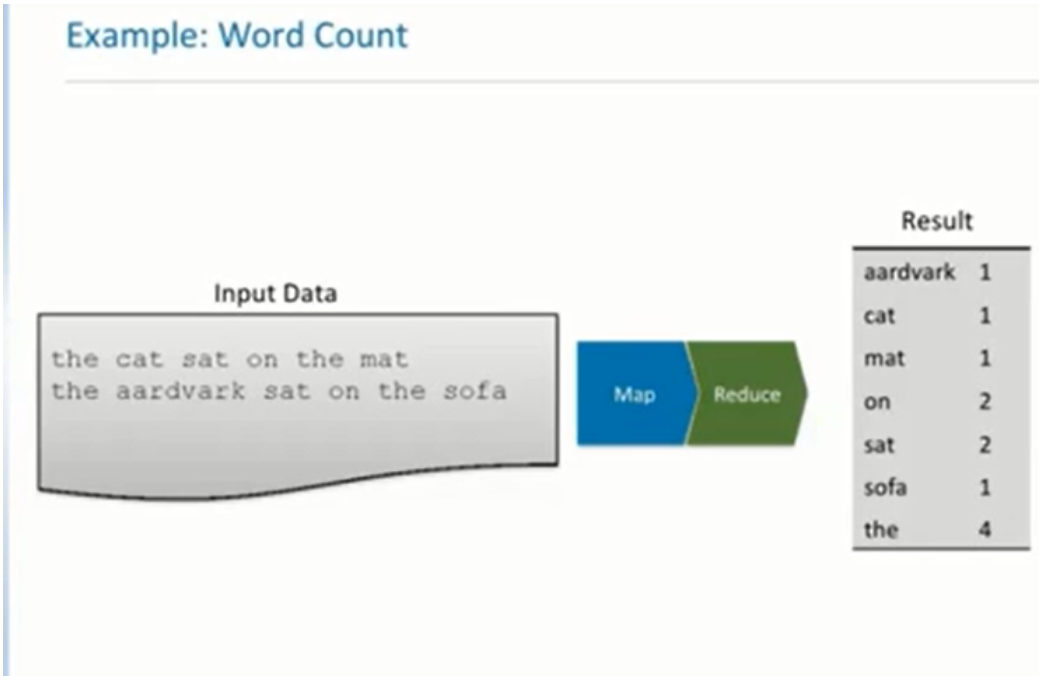

FIGURE 1.2 – *Exemple 1 : WordCount 21*

2. Il nous faut maintenant déterminer la clé à utiliser pour l'opération map. Le traitement classique et séquentiel pour résoudre ce problème va nous orienter vers le choix de prendre comme clés les mots du texte ;
3. L'étape suivante est d'écrire le code de l'opération map selon le schéma imposé par MapReduce, c'est-à-dire qu'elle doit retourner une liste de paires (clé, valeur) ;
4. Dans le cas de WordCount, l'opération map va donc décomposer le texte du fragment fourni en entrée et elle va générer pour chaque mot une paire (mot,1) ;
5. Nous avons donc maintenant tout ce qu'il faut pour l'étape MAP de MapReduce qui consiste à appliquer l'opération map à chaque fragment en parallèle comme l'illustre la figure 1.3et l'algorithme 1 06, 19
6. A la fin de l'étape MAP, nous avons donc plusieurs listes de paires (clé, valeur) ;
7. L'étape SHUFFLE and SORT permet de regrouper et de trier, par clé commune, les résultats intermédiaires fournis par l'étape MAP ;

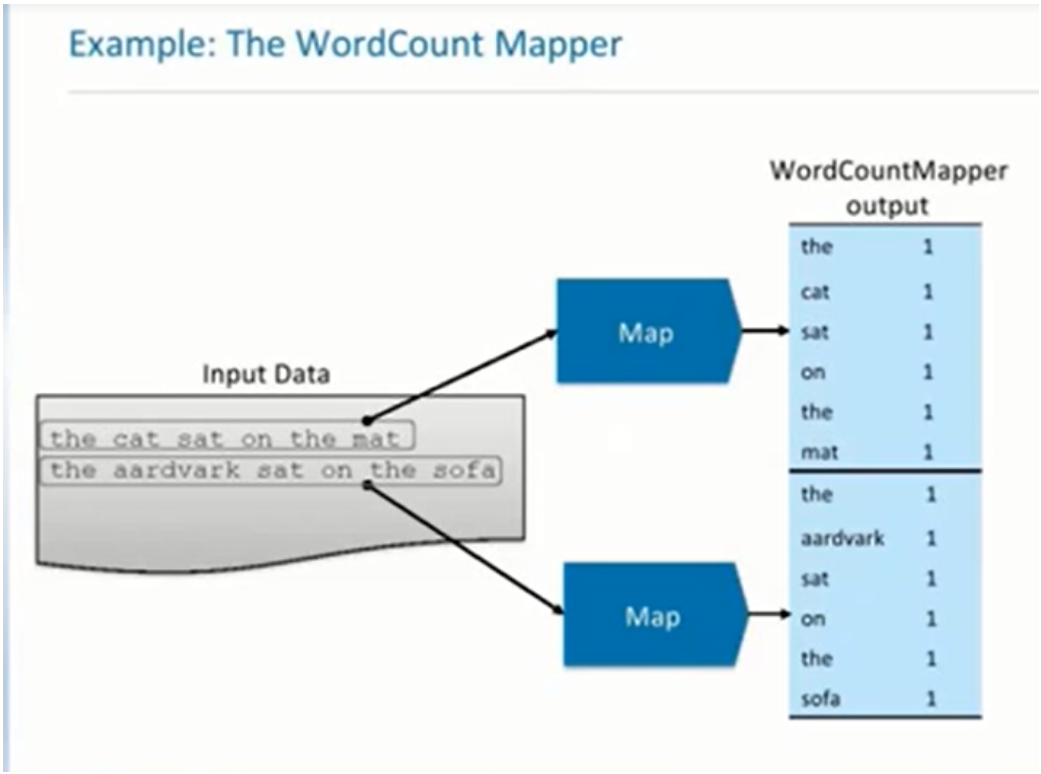

FIGURE 1.3 – *Liste des paires (clé, valeur)*

8. Cette étape est entièrement gérée par le framework d'exécution de MapReduce, de manière distribuée, figure 1.4 ;
9. Nous avons donc maintenant à notre disposition un ensemble de paires (clé, liste de valeurs).
10. Il nous reste maintenant à écrire le code de l'opération reduce, selon le schéma imposé par MapReduce. Pour WordCount, l'opération reduce va donc juste consister à sommer toutes les valeurs de la liste associée à une clé.
11. L'étape REDUCE de MapReduce peut donc être appliquée. Elle consiste à appliquer l'opération reduce à chaque paire (clé, liste de valeurs) en parallèle (voir l'algorithme 2).

1.3.4 Entre Map et Reduce : La phase Combine

On a vu que, les résultats intermédiaires sont traités dans une seule machine (machine maître) durant la phase Reduce, s'ils sont volumineux, le pro-

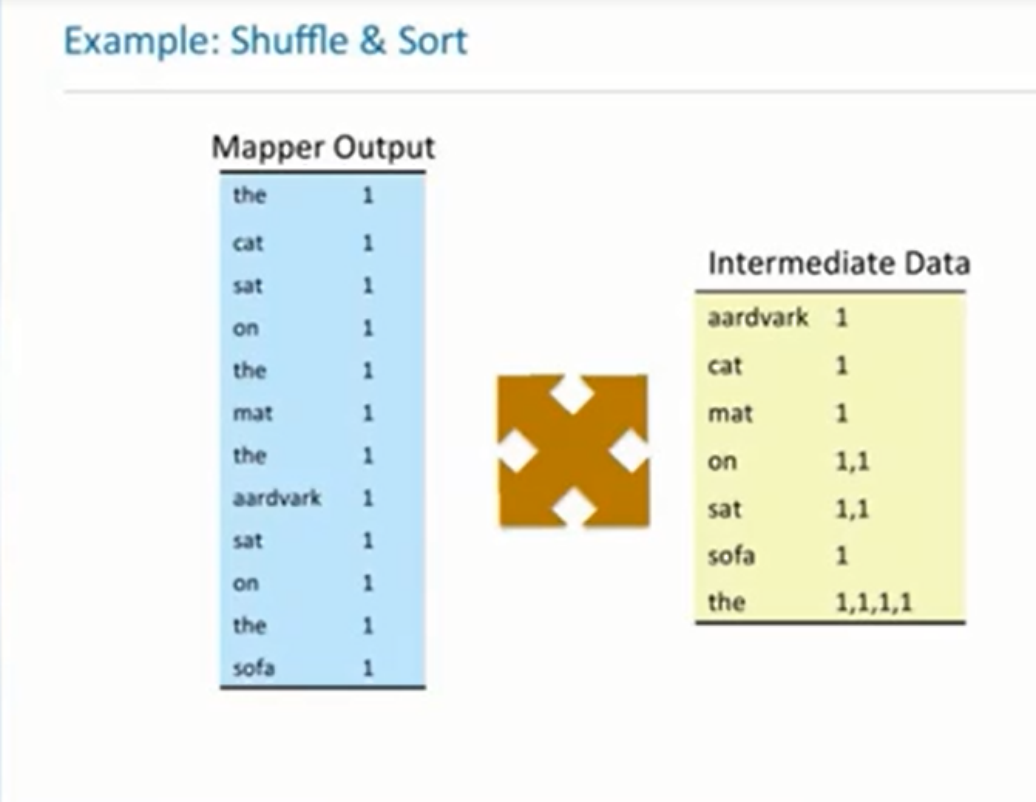

FIGURE 1.4 – *Etape SHUFFLE and SORT*

Algorithme 1 : Exemple : WordCount : fonction Map

```
fonction map(Entier cleEM, Texte valeurEM, Texte cleI, Entier valeurI);
// cleEM : est la clé en entrée du Map, le numéro du fragment,
// c'est le numéro de la ligne (l'offset)
// valeurEM : est la valeur en entrée du Map ou contenu
// du fragment, pour ce cas, c'est la ligne
pour chaque mot m dans valeurEM;
cleI := m;
valeurI :=1;
ecrire(cleI, valeurI).
```

Algorithme 2 : Exemple : WordCount : fonction Reduce

```
fonction reduce(Texte cleI, Liste d'Entier valeurI, Texte cleSR, Entier
 valeurSR);
// cleI : est la clé en entrée du Reduce, pour ce cas,
// c'est le mot (sortie du map)
// valeurI : est la liste des valeurs,
// pour ce cas, c'est une série de 1
cleSR := cleI;
wc :=0;
pour chaque valeur v dans valeurI wc+=v;
valeurSR :=wc;
ecrire(cleSR, valeurSR).
```

blème de lenteur réside toujours. Hadoop propose un troisième intervenant, entre map et reduce qui effectue un traitement local des paires produites par map. C'est la phase Combine. Son travail est de faire une première étape de réduction sur les résultats d'un map d'une machine. Il est surtout utilisé en mode distribué ou en mode psœudo distribué ou les deux en même temps comme le montre la figure 1.5 : .

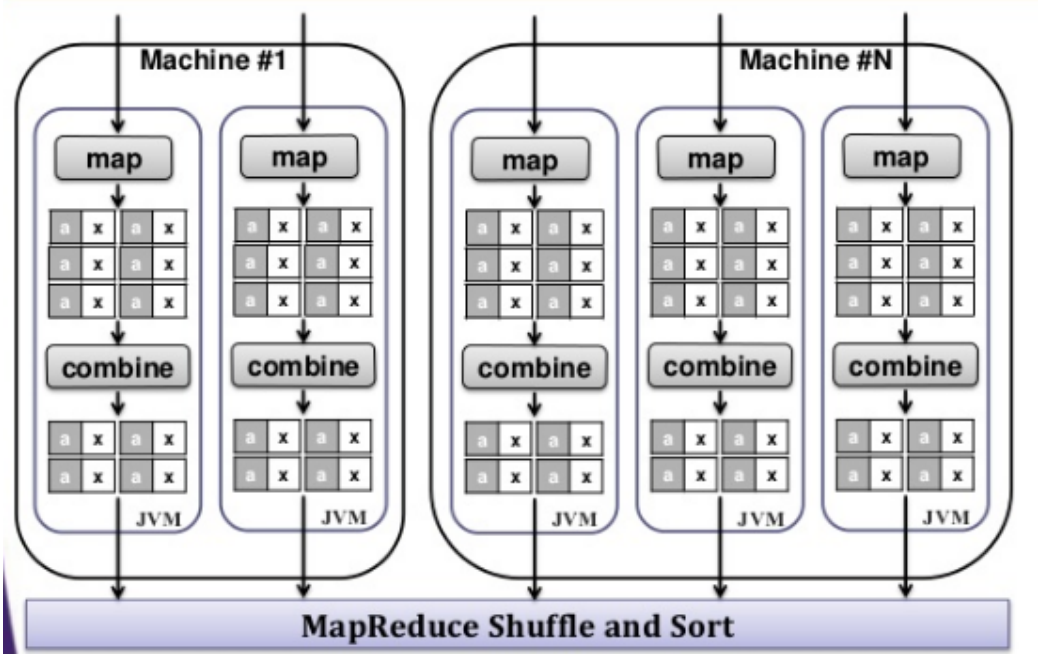

FIGURE 1.5 – *Schéma de la phase Combine 11*

— Il traite des paires ayant la même clé sur la même machine ou même JVM que les tâches Map. Les paires qu'il émet sont envoyées aux reducers.

— Le combiner permet de gagner du temps, en faisant le reduce sur chaque machine, les résultats obtenus sont intermédiaires, transmis à la machine maître pour le reduce final, exemple : pour calculer le maximum, au lieu que chaque machine envoi une liste de valeurs, combiner transmet une seule valeur qui est le maximum des valeurs dans une machine, c'est aussi valable pour le minimum, la somme et le produit,...

— Cette phase n'est pas valable dans tous les cas, autrement dit si l'opé-

rateur d'agrégation n'est pas commutatif ou pas associatif par exemple le calcul d'une moyenne, on ne peut pas intégrer l'étape "Combiner".

Différences entre Combine et Reduce

1. Les paramètres d'entrée et de sortie du Combiner doivent être identiques à ceux de sortie du Mapper, tandis que les types des paramètres de sortie du Reducer peuvent être différents de ceux de son entrée.
2. On ne peut pas employer un Combiner quand la fonction n'est pas commutative ou n'est pas associative.
3. Les Combiner reçoivent leurs paires d'un seul Mapper, tandis que les Reducers reçoivent les paires de tous les Combiners et/ou tous les Mappers. Les Combiners ont une vue restreinte des données.
4. Hadoop n'est pas du tout obligé de lancer un Combiner, c'est seulement une optimisation locale.

Wordcount avec Combiner

Reprenons l'exemple de Word Count, le psœudo algorithme de la fonction Map reste le même, celui de la fonction Combine est semblable à la fonction Reduce sauf que les sorties de Combine représentent une liste de résultats intermédiaires, tandis que le résultat fourni par la fonction Reduce est le résultat final, voir le psœudo algorithme 3, 4 illustré par la figure 1.6.

Algorithme 3 : Exemple : WordCount : fonction Combine

```
fonction combine(Texte cleI, Liste d'Entier valeurI, Texte cleI, Entier valeurI);
// cleI : est la clé en entrée du combine, pour ce cas,
// c'est le mot (c'est la sortie du map)
// valeuI : est la liste des valeurs,
// pour ce cas, c'est une série de 1, sortie du map
wcc :=0;
pour chaque valeur v dans valeurI wcc+=v;
valeurI := wcc;
ecrire(cleI, valeurI).
```

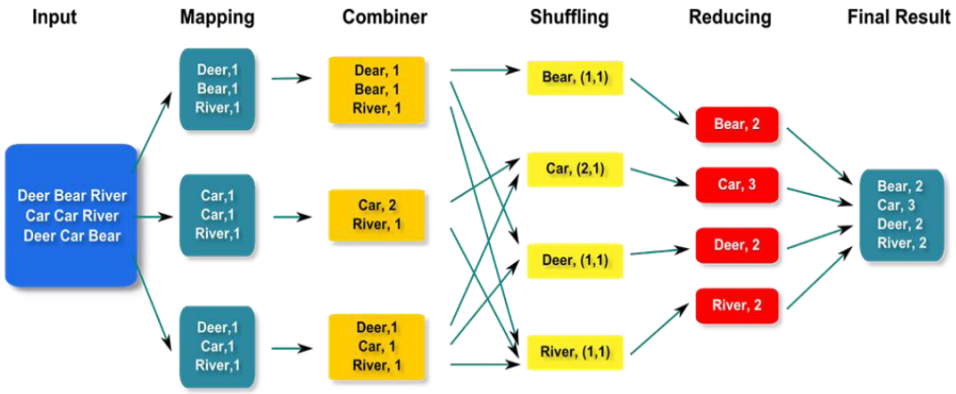

FIGURE 1.6 – *L'exemple WordCount avec Combiner* [12]

Algorithme 4 : Exemple : WordCount : fonction Reduce avec la phase combine

```
fonction reduce(Texte cleI, Entier valeurI, Texte cleSR, Entier valeurSR);
// cleI : est la clé en entrée du Reduce, pour ce cas,
// c'est le mot (sortie du combine)
// valeurI : est la liste des valeurs,
// pour ce cas, c'est une série de valeurs résultats de combine
cleSR := cleI;
wc :=0;
pour chaque valeur v dans valeurER wc+=v;
valeurSR :=wc;
ecrire(cleSR, valeurSR).
```

La modélisation des problèmes selon MapReduce est une tâche difficile et nécessite de l'entrainement, pour cela une série d'exercices avec solutions est proposée aux étudiants à la section 1.5.

1.4 MapReduce et Hadoop

La modélisation selon le paradigme MapReduce pour le traitement parallèle et distribué (section 1.3) consiste à formuler le problème à traiter en fonctions parallélisables tout en respectant les contraintes posées par le modèle.

Pour la concrétisation, on doit implémenter le modèle obtenu dans un contexte Big Data, pour cela, il faut savoir qu'il y a des tâches qui sont prises en charge par Hadoop (un framework d'exécution distribuée de MapReduce) et qui sont transparentes au développeur. Ces tâches concernent :

- l'ordonnancement et la distribution des traitements sur les différents nœuds du cluster ;
- l'accès et le partage de données à traiter ;
- la gestion des erreurs et la tolérance aux pannes ;
- la localisation des données à traiter.

1.4.1 MapReduce et Hadoop version 1.X

Rappelons que Hadoop est un framework d'une architecture maitre-esclave avec les concepts suivants (voir 06, 07, 08, 09) :

- Un Job MapReduce est une unité de travail que le client veut exécuter. Il consiste en trois choses :
  1. le fichier en entrée des données à traiter (Input file) ;
  2. le programme MapReduce ;
  3. les informations de configuration (Métadonnées).
- Le cluster exécute le job MapReduce en divisant le programme MapReduce en deux tâches : les tâches Map et les tâches Reduce.

— Dans le cluster Hadoop, il y a deux types de processus qui contrôlent l'exécution du job :

1. le jobtracker :
   — Le jobtracker est le processus maître, il est démarré sur le NameNode ;
   — Il coordonne tous les jobs qui s'exécutent sur le cluster ;
   — Il se charge de l'ordonnancement des traitements ;
   — Il gère les ressources du cluster ;
   — Il planifie l'exécution des tâches et les distribue sur les tasktrackers ;
   — Il reçoit (du client) la ou les tâches MapReduce à exécuter (un .jar Java) ainsi que les données d'entrée et le répertoire où stocker les données de sorties ;
   — Il est pour cela en communication avec le namenode d'HDFS ;
2. Un ensemble de tasktrackers :
   — Les tasktrackers sont des unités de traitement du cluster, ils assurent l'exécution et le suivi des tâches Map et Reduce s'exécutant sur son nœud ;
   — Ils sont démarrés au niveau des nœuds de données (Datanode), exécutent les tâche Map ou Reduce et envoient des rapports d'avancement au jobtracker, qui garde une copie du progrès de chaque job ;
   — Si une tâche échoue, le jobtracker peut la replanifier sur un tasktracker différent ;
   — Il dispose d'un nombre limité de slots d'exécution et donc un nombre limité de tâches MAP, REDUCE ou SHUFFLE pouvant s'exécuter simultanément sur le nœud ;
   — Il est aussi en communication constante avec le jobtracker pour l'informer de l'état d'avancement des tâches (heartbeat call). Car,

en cas de défaillance, le jobtracker, informé ou non du tastracker, doit pouvoir ordonner la ré exécution de la tâche.

Voici le schéma de soumission et d'exécution d'un job dans Hadoop MapReduce, figure 1.7 :

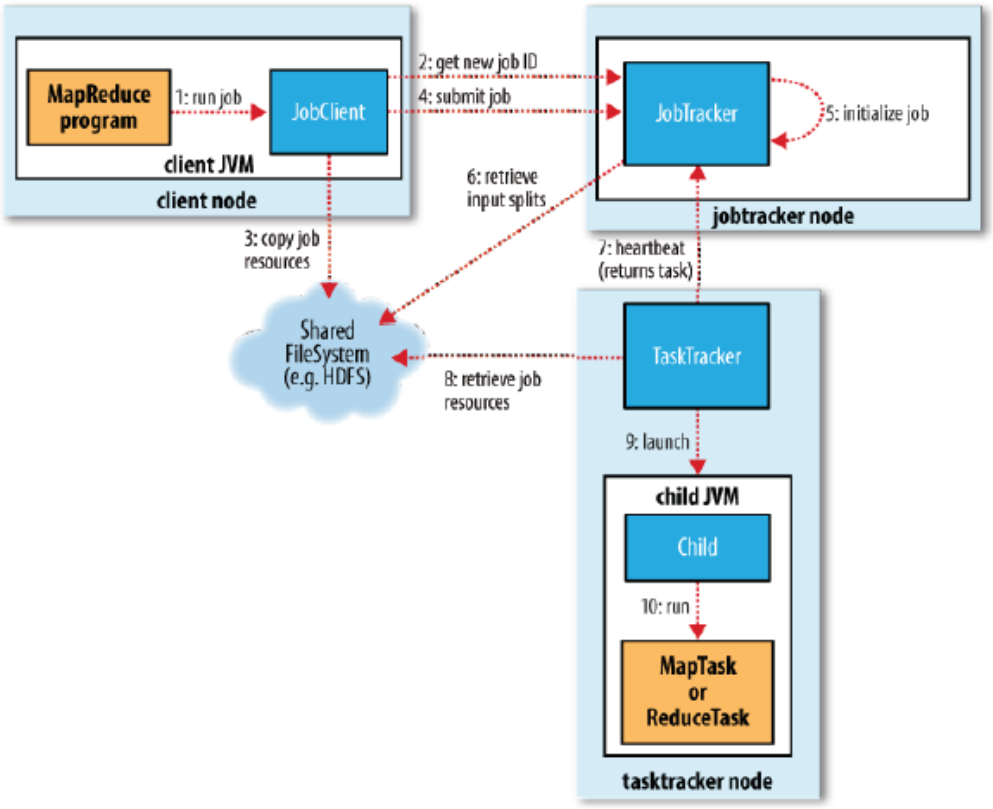

FIGURE 1.7 – *Comment Hadoop exécute un job MapReduce o8*

Explication du schéma, figure 1.7

1. Un client hadoop copie ses données sur HDFS ;
2. Le client soumet le job (travail) à effectuer au jobtracker sous la forme d'une archive.jar et des noms des fichiers d'entrée et de sortie (étape1) ;
3. Le processus de lancement de job demande aux JobTrackers la création d'un nouvel job ID avec getNewjobId() (étape2) ;
4. Il copie les ressources nécessaires pour exécuter le travail, y compris le fichier JAR (programme MapReduce) du job, les paramètres de configuration Hadoop (hdfs.xml et core-site.xml) et les sous-tâches. La copie se

fait dans le système de fichiers du jobTracker dans un répertoire nommé d'après l'ID du job. Le JAR du job est copié avec réplication selon le paramètre Mapred.submit.replication dans le fichier mapred-site.xml . (étape3)

5. Il indique au jobTracker que le job est prêt pour l'exécution en appelant submitJob() sur jobTracker (étape4) ;
6. L'exécution de submitjob() déclenche le processus d'initialisation du job avec l'étape 5 et 6
7. L'étape5 : Création d'un objet pour représenter le job ou la tâche à exécuter et lance un processus pour garder une trace de l'état d'avancement du job ;
8. L'étape6 : Récupération de la liste des sous tâches crées précédemment lors de l'étape3, récupération du nombre maximum de Reduce tasks configurées dans le fichier mapred-site.xml par la variable Mapred.Reduce.tasks.
9. Le processus affectation des tâches est déclenché en exécutant l'étape7 et 8 ;
10. Etape7 : les TaskTrackers envoient régulièrement des Heartbeat au jobTracker pour lui signifier leurs disponibilités à exécuter des jobs, si le jobtracker possède des jobs en file d'attente il confiera la tâche au TaskTracker ;
11. Etape8 : après attribution du job au TaskTracker, il commence par localiser le fichier JAR en le copiant depuis le système de fichiers partagés ;
12. TaskRunner lance une nouvelle JVM en (étape9) pour exécuter chaque tâche (étape10), c'est le processus "Exécution du job" Toutes ces tâches sont gérées par Hadoop et le développeur ne lui reste qu'à se concentrer sur :
    - L'écriture des programmes MAP et REDUCE et d'en faire une archive.jar ;

— La soumission des fichiers d'entrée, le répertoire de sortie et l'archive.jar au jobtracker.

1.4.2 MapReduce et Hadoop version2.x

On remarque que, de Hadoop version 1.X à Hadoop version 2.X avec YARN, la gestion des ressources est généralisée à d'autres applications que MapReduce et les fonctionnalités du jobtracker sont réparties entre 06 :

1. Le ResourceManager (RM) qui est le gestionnaire des ressources du cluster :
   — Il ordonnance les requêtes des clients et pilote le cluster par l'intermédiaire de node manager qui s'exécutent sur chaque nœud de calcul ;
   — Il a donc pour rôle de contrôler toutes les ressources du cluster et l'état des machines qui le constituent ;
   — Il gère donc le cluster en maximisant l'utilisation de ressources.
2. L'Application Master (AM) qui est un processus s'exécutant sur toutes les machines esclaves et gèrant, en discussion avec le manager, les ressources nécessaires au travail soumis.

De même, les fonctionnalités du tasktracker sont aussi réparties sur une même machine entre :

— des contraintes qui sont des abstractions de ressources sur un nœud dédiées soit à l'exécution de tâches comme Map et Reduce, soit à l'exécution d'une application master.
— node manager qui héberge des containers et gère donc les ressources du nœud. Il est en communication via un heartbeat avec le ressource manager.

Le schéma de soumission et d'exécution d'un job dans cette nouvelle architecture est donc le suivant, (figure 1.8) :

1. Un client contacte le ResourceManager et lui demande d'exécuter un processus principal d'une application (étape1) ;

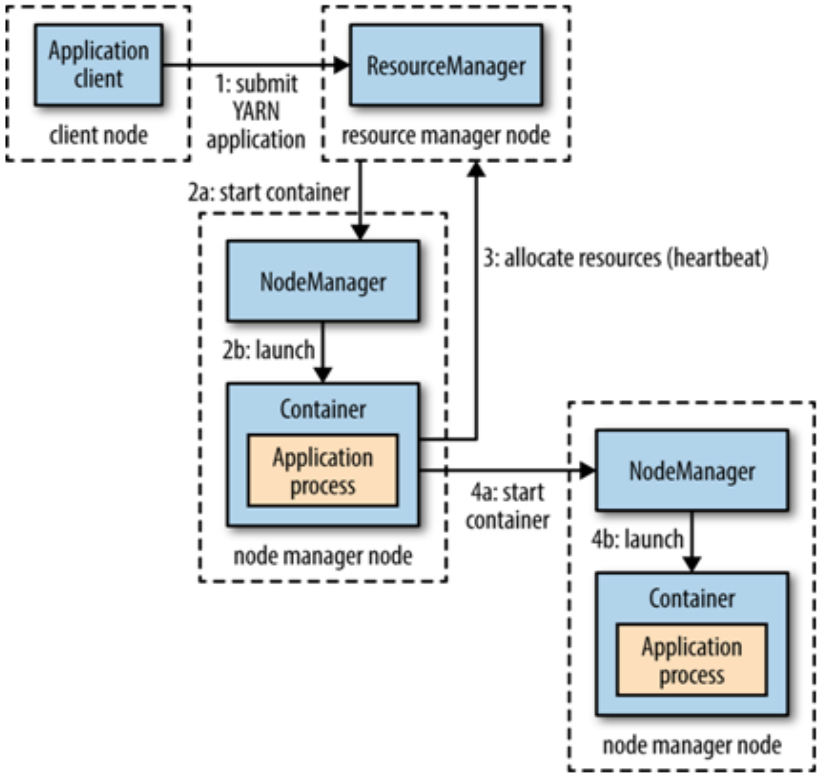

FIGURE 1.8 – *Schéma simplifié de l'exécution d'un travail dans Hadoop 2.X avec YARN* [20]

2. Le Manager trouve ensuite un gestionnaire de nœud (NodeManager) pouvant lancer l'AM (maître d'application) dans un conteneur (container) (étape 2a et 2b) ;
3. Ce que l'AM fait, une fois lancé, dépend de l'application. Il peut simplement exécuter un calcul dans le conteneur dans lequel il s'exécute et renvoyer le résultat au client.
4. Il y a la possibilité de demander plus de conteneurs au RM (étape 3) ;
5. Et les utiliser pour exécuter un calcul distribué (étapes 4a et 4b)

1.5 Exercices

1.5.1 Exercice1 :

Une entreprise de téléphone veut calculer la durée totale des appels téléphoniques des abonnés à partir d'un fichier contenant tous les appels (n°abonné, n°appelé, date, durée d'appel).
Question : Ecrire le psœudo algorithme des fonctions Map et Reduce pour le traitement de ce problème.

1.5.2 Exercice2 :

Question1 : Ecrire les fonctions Map et Reduce qui donnent la longueur moyenne par ligne des mots dans un texte.
Exemple :
algorithme, système
fichier, mapreduce
On aura comme résultat :

1. pour la ligne1 : longueur $= 17$, nombre de mots $= 2$, longueur moyenne $= 17/2$
2. longueur $= 16$, nombre de mots $= 2$, longueur moyenne $= 16/2$

Question2 : Ecrire les fonctions Map et Reduce qui donnent la longueur moyenne de tous les mots dans un texte.
longueur $= 33$, nombre de mots $= 4$, longueur moyenne $= 33/4 = 8,25$

1.5.3 Exercice3 :

Nous disposons d'un fichier de données volumineux qui contient les ventes annuelles des produits. Chaque ligne contient (l'année, le code produit "ID-produit" et la vente annuelle du produit). Donc, chaque année apparaît avec une liste des ventes. (voir un extrait du fichier ci-dessous)

Question1 : On souhaite effectuer un traitement parallèle selon le paradigme MapReduce afin d'afficher pour chaque année, la vente la plus élevée (vente maximale par année). Ecrire l'algorithme des fonctions Map() et Reduce().

Année, ID-produit, Vente

2015, 01, 102

2016, 02, 305

2017, 01, 429

.....

Question2 : Sachant qu'on dispose d'un autre fichier qui contient les détails produits (ID-produit, Intitulé), voir ci-dessous. On veut afficher l'intitulé de chaque produit avec la somme totale des ventes. Pour cela, il faut faire la jointure du fichier vente avec le fichier Produit, calculer et afficher la somme des ventes par Intitulé de produit.

ID-produit, Intitulé

01, table

02, chaise

.....

Annexes

A

Bibliographie

[01]. site : Le big data, 2019. URL https://www.lebigdata.fr/definition-big-data.

[02]. Stephane Vialle, CentraleSuplec. Big Data : Informatique pour les donnees et calculs massifs. 24 mai 2017 .

[03]. mohammed zuhair al taie. hadoop ecosystem : an integrated environment for big data. in big data, guest posts 2015. URL http://blog.agroknow.com/?p=3810.

[04]. site : Hadoop. 2016. URL http://www.opentuto.com/category/web-2/big-data/hadoop/.

[05]. amal abid. cours big data : Chapitre2 : Hadoop. 2017. URL https://fr.slideshare.net/AmalAbid1/cours-big-data-chap2.

[06]. celine hudelot, regis behmo. realisez des calculs distribue sur des donnees massives. 2019. URL https://openclassrooms.com/fr/courses/4297166-realisez-des-calculs-distribues-sur-des-donnees-massives. (Cité pages 2, 4, 9, 15 et 19.)

[07]. Benaouda, Sid Ahmed Amine, implantation du modele mapreduce dans l'environnement distribue, 2015 . URL http://dspace.univ-tlemcen.dz. (Cité page 15.)

[08]. Tom White, Hadoop : The Definitive Guide, June 2009 : First Edition. (Cité pages 15 et 17.)

[09]. Srinath Perera, Thilina Gunarathne, Hadoop MapReduce Cookbook, First published : February 2013. (Cité page 15.)

[10]. Pierre Nerzic, Outils Hadoop pour le BigData, mars 2018.

[11]. marty hall. map reduce 2.0 (input and output). 2013. URL https://www.slideshare.net/martyhall/hadoop-tutorial-mapreduce-part-4-input-and-output. (Cité page 12.)

[12]. build projects, learn skills, get hired. hadoop mapreduce- java-based processing framework for big data. URL https://www.dezyre.com/hadoop-course/mapreduce. (Cité page 14.)

[13]. URL https://hadoop.apache.org/docs/r2.4.1/api/org/apache/hadoop/mapreduce/.

[14]. URL https://gist.github.com/kzk/712029/9d0833aac03b23ec226e034d98f5871d9580724e.

[15]. Jeffrey Dean and Sanjay Ghemawat, MapReduce : Simplified Data Processing on Large Clusters, 2004 . (Cité page 2.)

[16]. Univ. Lille1- Licence info 3eme annee, Cours nÂ°1 : Introduction Ã la programmation fonctionnelle, 2013-2014 . (Cité page 3.)

[17] dataflair team. hadoop architecture in detail hdfs,yarn et mapreduce. 2019. URL https://data-flair.training/blogs/hadoop-architecture/. (Cité page 7.)

[18] vlad korolev. hadoop on windows with eclipse. 2008. URL http://v-lad.org/Tutorials/Hadoop/.

[19]. Donald Miner and Adam Shook. MapReduce Design Patterns. OReilly, 2012. (Cité page 9.)

[20]. Tom White. Hadoop : The Definitive Guide, 4th Edition. OReilly, 2015. (Cité page 20.)

[21]. Hadoop Training. 2018. URL `https://www.slideshare.net/AnandMHadoop/session-19-mapreduce`. (Cité page 9.)