Université Mohammed V- Rabat
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique
Filière Génie Informatique et Digitalisation

Fondements du Big Data

Pr. N. EL FADDOULI
nfaddouli@gmail.com

2023-2024

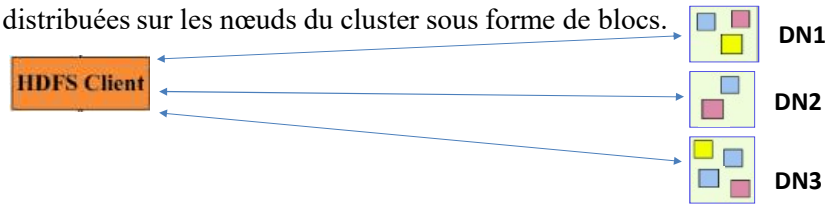
CC-BY NC SA

MapReduce/YARN

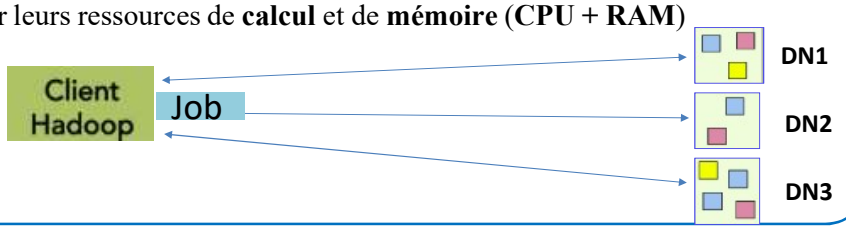
Hadoop: MapReduce

- **Principe:**

- Les données sont distribuées sur les nœuds du cluster sous forme de blocs.



- Ces données ne seront pas déplacées, via le réseau, vers un programme devant les traiter.
- Pour de meilleures performances, chaque bloc de données est traité **localement** *(principe de **data locality**)*, minimisant les besoins d'échanges réseaux entre les machines.
- Un programme (*job*) doit donc être exécuté sur les nœuds contenant les données à traiter, où il va exploiter leurs ressources de **calcul** et de **mémoire** (**CPU + RAM**)

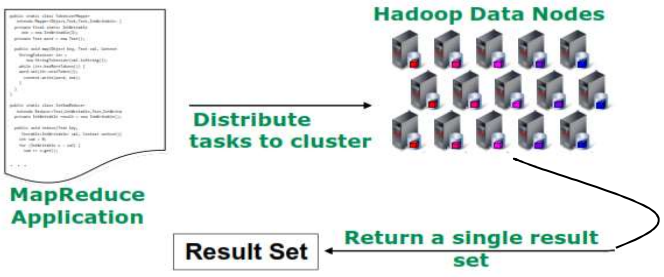


 CC-BY NC SA

Hadoop: MapReduce

- **Principe:**

- Le programme doit être écrit selon un **modèle** bien déterminé: **MapReduce**.
- Le système de fichiers distribués (HDFS) est au cœur de **MapReduce**. Il est responsable de la distributions des données à travers le cluster, en faisant en sorte que l'ensemble du cluster ressemble à un système de fichiers géant.



 CC-BY NC SA

Hadoop: MapReduce

• **Intérêt:**

- Simplifier le développement de programme devant traiter des données distribuées.
- Le développeur n'a pas à se soucier du travail de **parallélisation** et de **distribution** du **traitement (*job*)**. **MapReduce** permet au développeur de ne s'intéresser qu'à la partie **algorithmique** à appliquer aux données.
- Un programme **MapReduce** contient deux fonctions principales **Map ()** et **Reduce ()** contenant les traitements à appliquer aux données.

• **Architecture MR1 (dans Hadoop 1)**

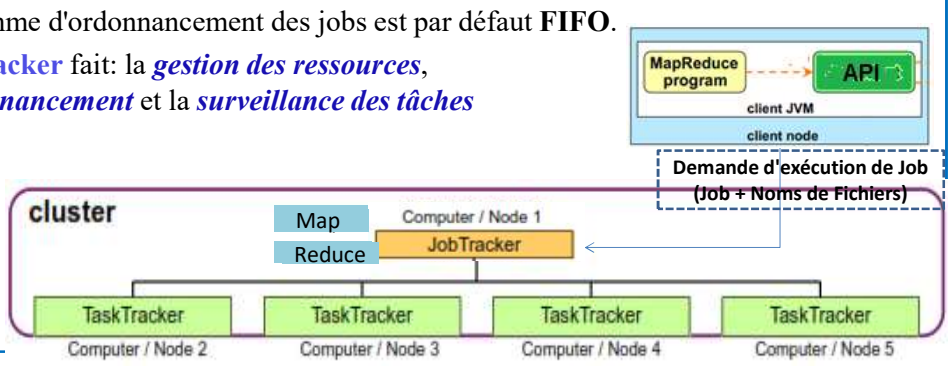
- **Maître/Esclave**: L'unique maître (**JobTracker**) contrôle l'exécution des deux fonctions sur plusieurs esclaves (**TaskTrackers**).

 CC-BY NC SA

MapReduce: JobTracker

- Reçoit les **jobs MapReduce** envoyés par les clients (*Applications*)
- Communique avec le **NameNode** pour avoir les emplacements des données à traiter
- Passe les tâches **Map** et **Reduce** aux nœuds **TaskTrackers**
- Surveille les tâches et le statut des **TaskTrackers**
- Relance une tâche si elle échoue.
- Surveille l'état d'avancement des jobs et fournit des informations à ce sujet aux applications clientes.
- Algorithme d'ordonnancement des jobs est par défaut **FIFO**.

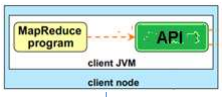
➔ **JobTracker** fait: la ***gestion des ressources***, l'***ordonnancement*** et la ***surveillance des tâches***

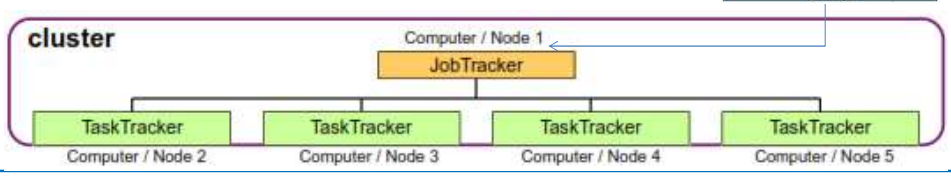


 CC-BY NC SA

MapReduce: TaskTracker

- Exécute les tâches **Map** et **Reduce**
- Chaque TaskTracker est configuré avec un nombre de "**slots**", qui représentent sa le nombre de tâches **Map** et **Reduce** qu'il peut exécuter (**mapreduce.tasktracker.map.tasks.maximum** et **mapreduce.tasktracker.reduce.tasks.maximum** dans **mapred-site.xml**).
- Pour lancer une tâche Map, le JobTracker cherche un slot Map libre sur un nœud possédant les données à traiter. S'il n'existe pas, il cherche sur un nœud du même rack.
- Communique son statut au **JobTracker** via des **heartbeat** (*id, slots libres*, …)
- Après une durée de *mapreduce.jobtracker.expire.trackers.interval* (=10 min par défaut) le TaskTracker est considéré comme perdu.
- Gère le stockage des sorties intermédiaires



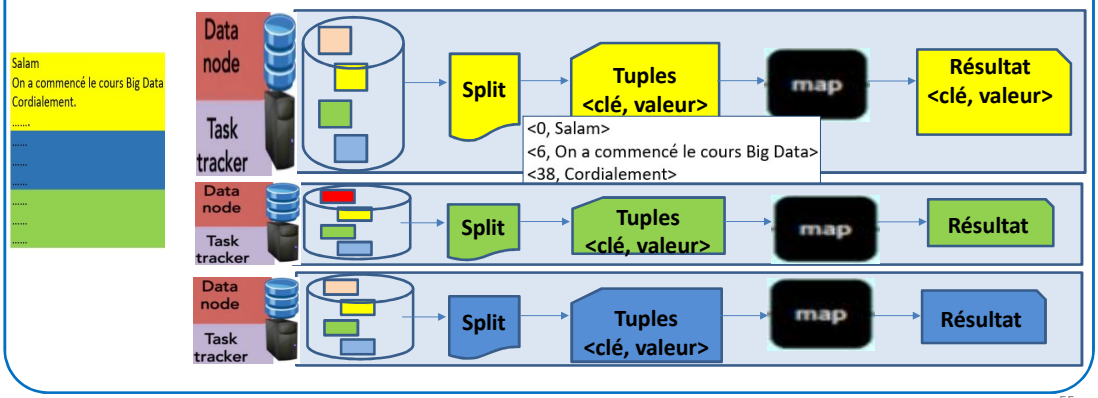


54

FONDEMENTS DU BIG DATA \ N.EL FADDOULI CC-BY NC SA

MapReduce: Modèle de programmation - Map

- Les fichiers d'entrée sont divisés en pièces **logiques** appelées "**Input Splits**" (**Splits**)
- Un **split** est une division **logique** des fichiers qui sont stockés sous forme de **blocs** HDFS. Ces blocs représentent une division **physique** des fichiers.
- Un **split** **désigne** une partie d'un fichier (*début*, *fin*, *id blocs+emplacments*, …)
- La taille d'un **split** peut être définie dans le job à exécuter (*par défaut: taille de bloc*) (max(mapred.min.split.size, min(mapred.max.split.size, dfs.block.size))

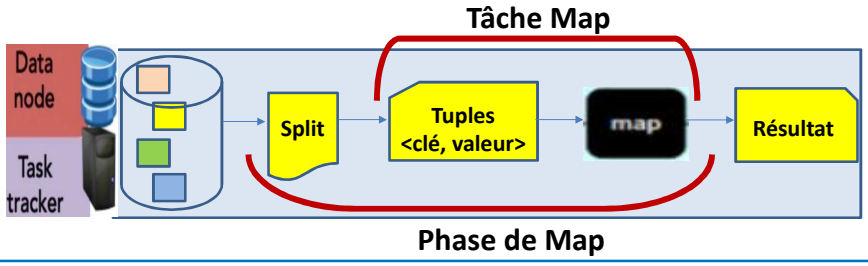


55

FONDEMENTS DU BIG DATA \ N.EL FADDOULI CC-BY NC SA

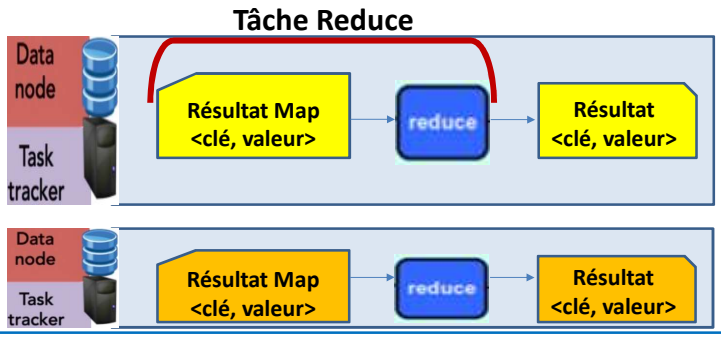
MapReduce: Modèle de programmation - Map

- Les données (lignes) correspondantes à un split sont transformées en tuples *<Clé, Valeur>*
- Chaque **split** est traité par une **tâche Map** qui applique la fonction **Map** de l'utilisateur sur chaque **tuple** du split.
- Les nœuds esclaves (**TaskTracker**) exécutent les tâches Map pour traiter les **splits** individuellement en parallèle (sous le contrôle global du **JobTracker**)
- Chaque nœud esclave stocke les résultats de ses tâches Map dans son **système de fichiers local** s'ils dépassent un seuil dans la RAM.



MapReduce: Modèle de programmation - Reduce

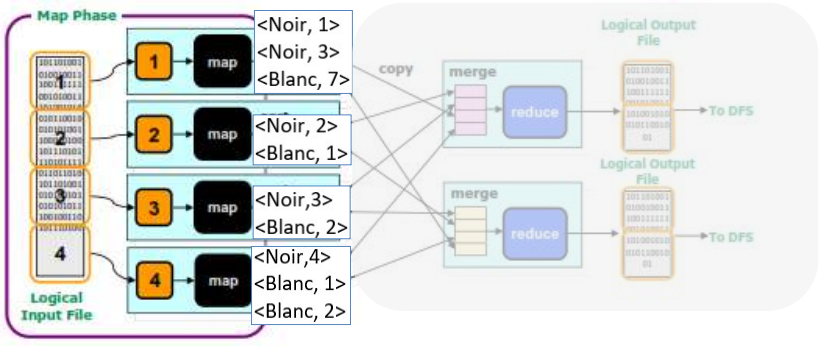
- Les résultats des tâches **Map** sont agrégées par des **tâches Reduce** sur des nœuds esclaves (sous le contrôle du **JobTracker**).
- Plusieurs **tâches Reduce** parallélisent l'agrégation
- Une **tâche Reduce** appliquent le fonction **Reduce** de l'utilisateur sur les résultats (**<clé, valeur>**) de tâches **Map**.
- Les sorties sont stockées dans **HDFS** (**et donc répliquées**)



MapReduce: La phase de Map

Mapper

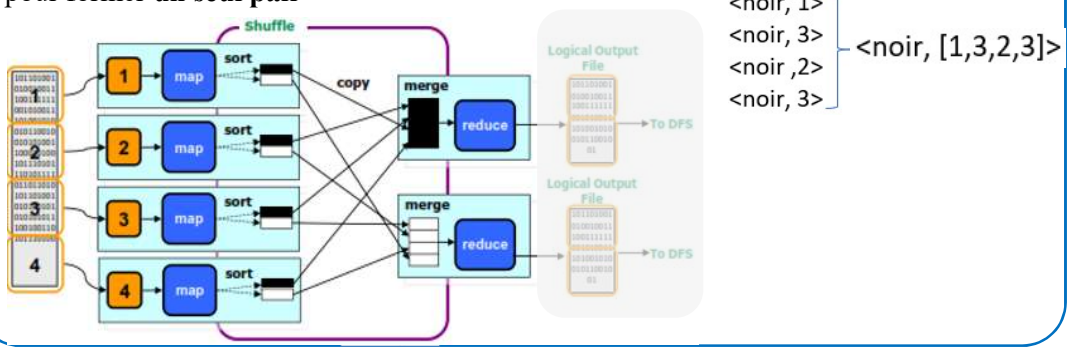
- Petit programme (*généralement*), distribué dans le cluster et local aux données
- Traite une partie des données en entrée (*Split*)
- Chaque **Map analyse**, **filtre** ou **transforme** un split qui est un ensemble de pairs **<key, value>.**
- Produit des pairs **<clé, valeur>** groupés



 CC-BY NC SA

MapReduce: La phase de Shuffle (réorganisation)

- Le traitement de cette phase est **préprogrammé** dans **MapReduce**
- Le résultat ({***<clé, valeur>***}) produit par chaque **Map** est **localement regroupée** par clé.
- Les pairs de **même clé** sont envoyés (**copiés**) au **même nœud**, choisi par le **JobTracker,** qui exécutera la phase **Reduce** sur ces données.
- Avant l'exécution de la tâche **Reduce**, les pairs de même clé sont **fusionnés** sur ce nœud pour former **un seul pair**

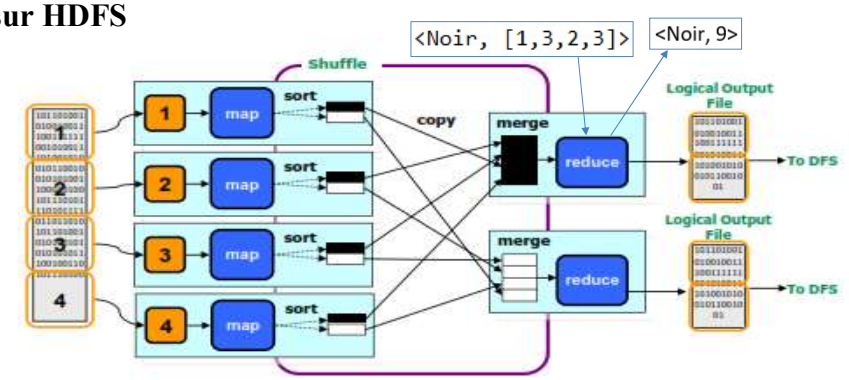


 CC-BY NC SA

MapReduce: La phase de Reduce

Reducer

- Petit programme (généralement) qui traite toutes les valeurs de la clé dont il est responsable. Ces valeurs sont passées au **Reducer** sous forme d'un tableau.
- Chaque **Tâche Reduce** écrit son résultat **dans son propre fichier de sortie sur HDFS**

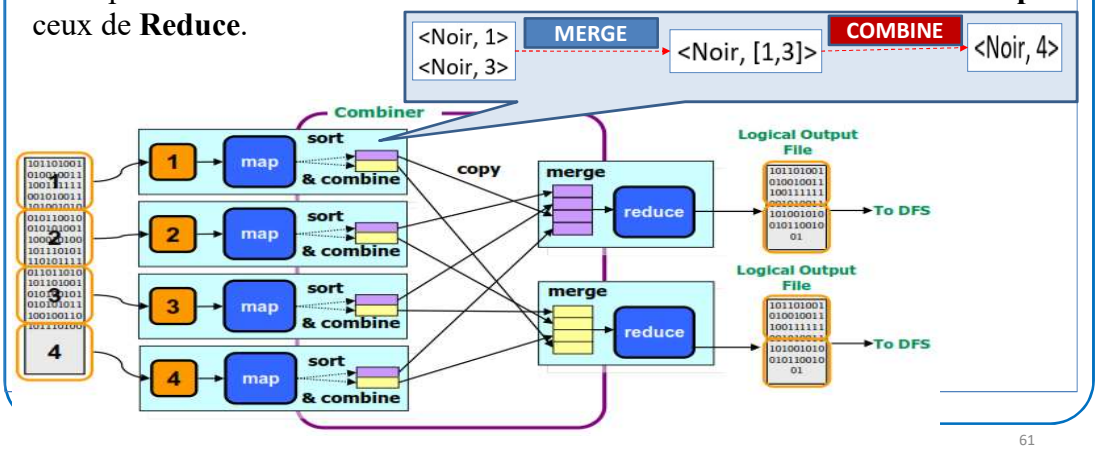


 CC-BY NC SA

MapReduce: La phase de Combine *(optionnel)*

Combiner

- Les résultats de **Map** sont **triés** et **fusionnés (par clé)** en local puis traités par une fonction généralement identique à **Reduce** exécutée par les nœuds ayant réalisé l'opération de **Map**.
- Ceci permettra aussi de **minimiser le trafic réseau** entre les nœuds de **Map** et ceux de **Reduce**.



 CC-BY NC SA

Exemple MapReduce

Objectif: On a un fichier texte sur HDFS contenant les noms d'animaux (*un nom/ligne*) qui se répètent.

On veut calculer le nombre de chaque félin (*tigre*, *lion*, *panthère*, ….)

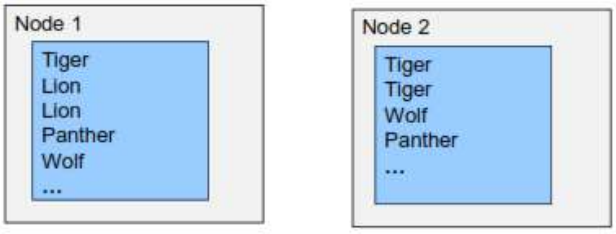
• En SQL, on aurait pu avoir le résultat par la requête:

```
SELECT NAME, COUNT(NAME) FROM ANIMALS
WHERE NAME IN ('Tiger', 'Lion', …)
GROUP BY NAME;
```

Table

Animals (Name Varchar2(20), …..)

• Le fichier a été divisé, par exemple, en deux blocs sur deux nœuds:



Comme les deux blocs sont sur deux nœuds différents, ils seront traités séparément.

Ex… La tâche Map

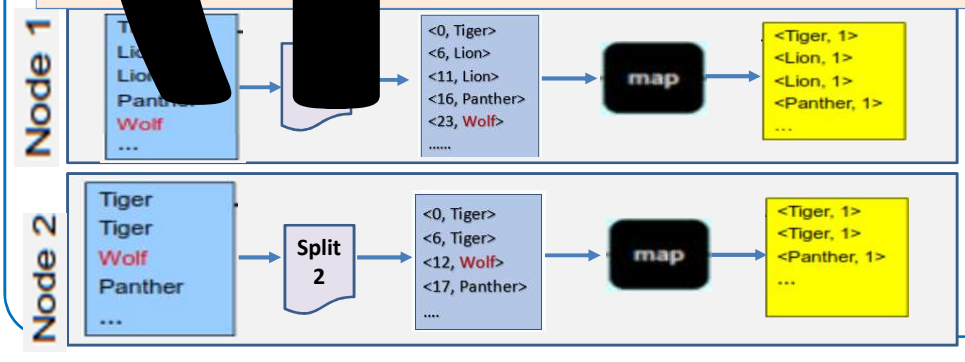
• M… …chier aux sauts de ligne

• Filtr… …*<clé, valeur>* comme suit: ***<nom, 1>***

• La tâ… …**plit** localement (*sur les nœuds qui hébergent le(s) bloc(s)*

• Si les … …ud devant traiter le split, elles lui seront envoyées depuis … …*ment sur le même rack*)

```
pub… …value, Context sortie) { String V = value.toString();
                    …rite(value, 1) ; }
```



MapReduce: Code de base de la tâche Map

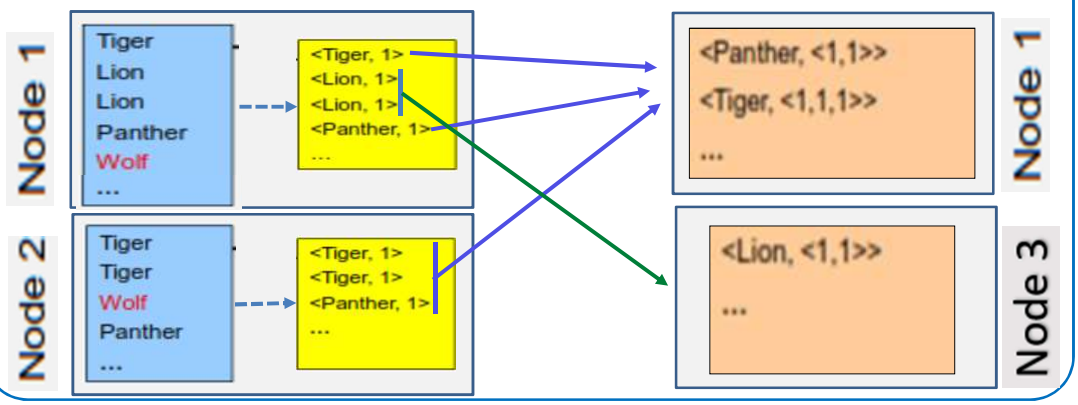
Voir TP

```
public class MyMapper  extends Mapper<LongWritable, Text, Text,  IntWritable>
{
 ..................................... // attributs
 @Override
  public void map(LongWritable key, Text   value, Context    context)
                    throws IOException, InterruptedException
          {
           ..... // traitement de value
          context.write(  new Text(...),   new IntWritable(...)  ); // génération d'un pair
          }
 }
```


 CC-BY NC SA

Exemple MapReduce: La tâche Shuffle

- Déplacer tous les **pairs ayant la même clé** vers le **même nœud** cible
  clé[0] ∈ ['A', 'L'] → pairs vers **Noeud3**, clé[0] ∈ ['M', 'Z'] → pairs vers **Noeud1**
- Les tâches **Reduce** peuvent s'exécuter sur n'importe quel nœud.
- Le nombre de tâches **Map** et **Reduce** n'est pas forcément le même.
- Généralement le nombre de tâches **Reduce** est plus petit que celui des **Map**.

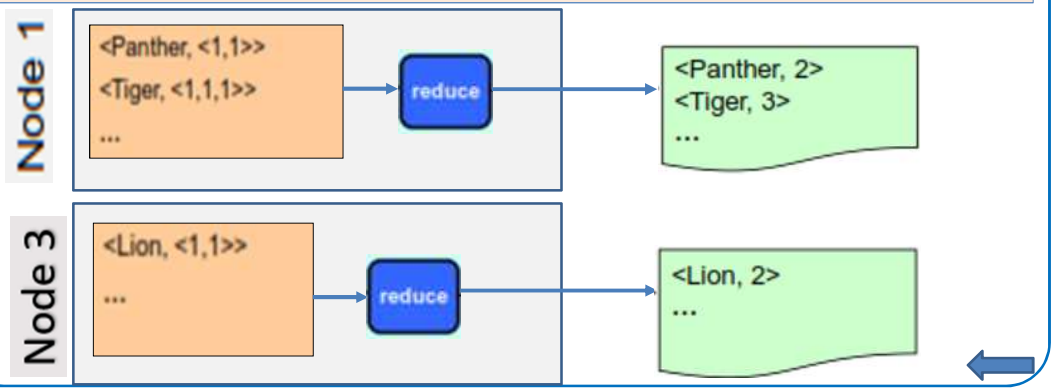


 CC-BY NC SA

Exemple MapReduce: La tâche Reduce

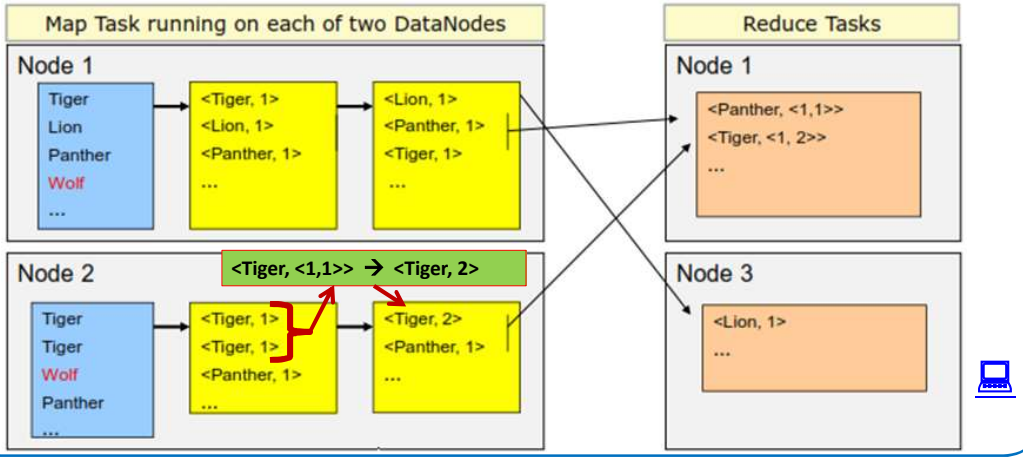
- La tâche **Reduce** calcule des valeurs agrégées pour chaque clé, dans notre cas, le nombre d'occurrence de la clé (*nom d'animal*) c'est ***la somme des valeurs fusionnées***.
- La sortie (résultat) d'une tâche **Reduce** est écrite dans un fichier HDFS

```
public void reduce(Text key, Iterable<IntWritable> values, Context sortie) { int sum = 0;
for (IntWritable V: values) { sum += V.get(); } sortie.write(key , new IntWritable(sum)); }
```



Exemple MapReduce: La tâche Combine

- Pour améliorer les performances, les résultats de **Map** sont **agrégées** sur les **nœuds l'ayant produit** (**avant** la phase **Shuffle**)
- On réduit la quantité de données envoyées sur le réseau



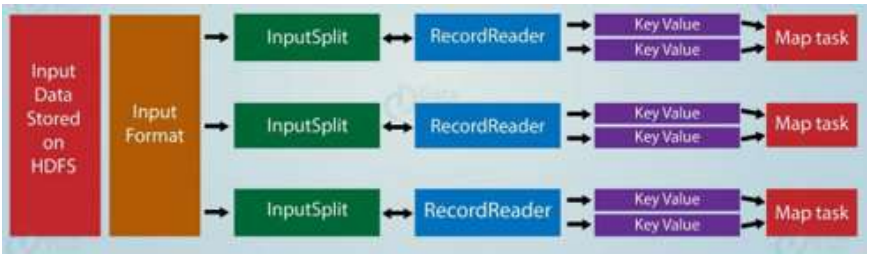
MapReduce: SPLITS

- Les fichiers dans Hadoop sont stockés dans des blocs (128 Mo).
- **MapReduce** divise logiquement les données en fragments ou **Splits**.
- Une tâche **Map** est exécutée sur chaque **Split**.
- La plupart des fichiers sont sous forme d'enregistrements séparés par des caractères bien définis.
- Le caractère le plus courant est le caractère de fin de ligne.
- La classe **InputFormat** est chargée de prendre un fichier HDFS et le transformer en **Splits** (**InputSplit**). (La méthode **getSplits** retourne une liste d'objet **InputSplit** dont chacun représente un split <*chemin du fichier, début, fin, …*>)

 CC-BY NC SA

MapReduce: Quelques Classes

Trois classes principales lisent des données dans MapReduce:

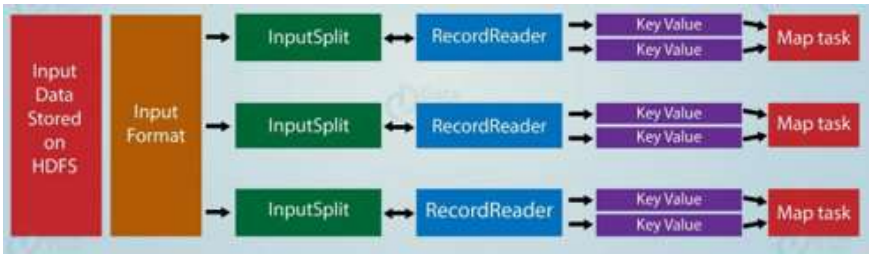


InputFormat: divise les fichiers à traiter en plusieurs **splits** dont la taille est **128M** ou la valeur du paramètre ***mapred.min.split.size*** dans ***mapred-site.xml***. On peut aussi préciser la taille dans le code du job MapReduce. On peut également créer une classe **InputFormat** personnalisée pour indiquer comment un fichier peut être divisé en **splits**.

InputSplit: Chaque split est représenté par un objet de la classe **InputSplit** qui est une représentation logique des données. Les données traitées par une tâche **Map** sont représentées par un objet **InputSplit** (*taille, informations sur les nœuds contenant les données du split*)

 CC-BY NC SA

MapReduce: Quelques Classes

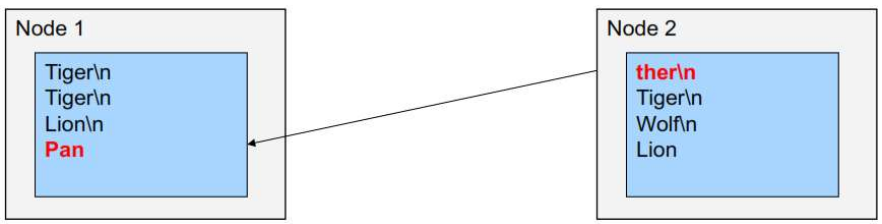


- **RecordReader**: Elle communique avec **InputSplit** pour convertir les lignes (**LineRecordReader**) des fichiers (**Splits**) en pairs ***<clé, valeur>.*** Par défaut, **RecordReader** utlise **TextInputFormat** pour faire cette conversion.

 CC-BY NC SA

MapReduce: RecordReader

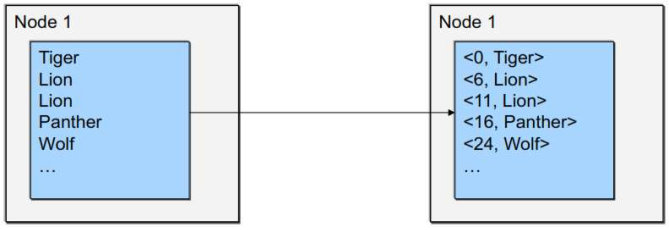
- La plupart du temps, la fin d'un split ne se trouve pas à la fin d'un bloc
- Les fichiers sont lus dans des **Records** par la classe **RecordReader**.
- Dans le cas où le dernier enregistrement d'un bloc se termine dans autre bloc, HDFS enverra la partie manquante du dernier enregistrement via le réseau



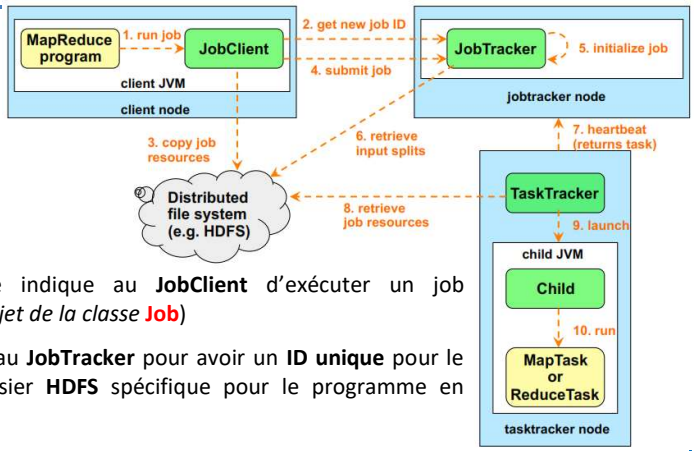
 CC-BY NC SA

MapReduce: InputFormat

- Les tâches **MapReduce** lisent les fichiers en définissant une classe **InputFormat**.
  - Les tâches **Map** attendent des pairs ***<clé, valeur>***
- Pour lire des fichiers texte de lignes délimitées, Hadoop fournit la classe **TextInputFormat**.
- Elle retourne un pair ***<clé, valeur>*** par ligne
- La **valeur** est le contenu de la ligne
- La **clé** est le décalage par rapport au début de la première ligne.

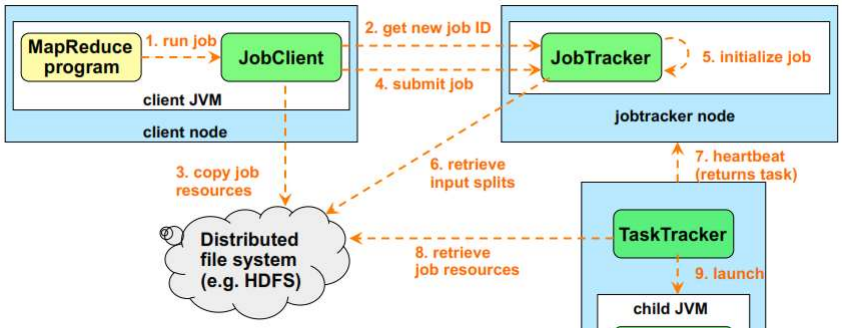


MapReduce: Etapes d'exécution d'un job



1. Le programme **MapReduce** créé indique au **JobClient** d'exécuter un job **MapReduce** (*via l'utilisation d'un objet de la classe* **Job**)
2. Le **JobClient** envoie une demande au **JobTracker** pour avoir un **ID unique** pour le job. Le **JobTracker** créera un dossier **HDFS** spécifique pour le programme en utilisant son **ID**.
3. Le **JobClient** calcule le nombre de splits du job et copie, dans le dossier **HDFS** créé par le **JobTracker** et nommé d'après l'**ID du job**, les ressources du job, telles qu'un fichier **jar** contenant le code Java des tâches **Map** et **Reduce,** un fichier XML de configuration du *job (classe de Map et Reduce, type de clé et valeur des résultats de Reduce, …)* et les splits calculés.
   Le JAR du job est copié avec une réplication haute (*paramètre* **mapreduce.submit.replication** dans **mapred-site.xml**, par **défaut=10**), de sorte qu'il y est beaucoup de copies à travers le cluster pour les **TaskTrackers**.

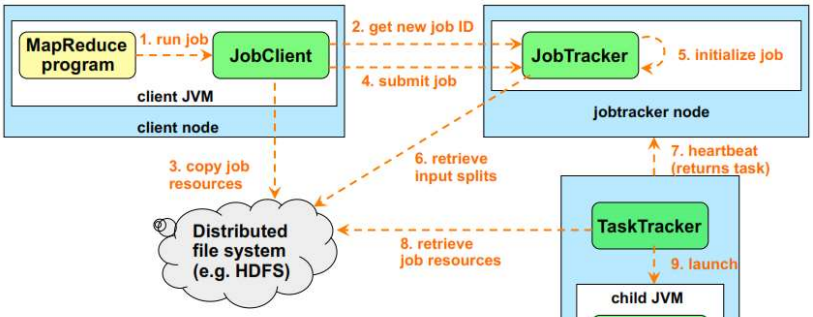
MapReduce: Etapes d'exécution d'un job



4. Une fois que les ressources sont dans **HDFS**, le **JobClient** peut demander au **JobTracker** de démarrer l'exécution du job.
5. Le **JobTracker** effectue l'initialisation job. Il crée un objet représentant le job à exécuter et lance un processus pour garder trace de son état d'avancement.
6. Il récupère: (**1**) les splits du dossier HDFS du job afin d'avoir le nombre de tâches **Map,** (**2**) le nombre maximal des tâches **Reduce** à créer (paramètre **Mapred.Reduce.tasks** dans **mapred-site.xml, par défaut=1** càd 100% de la taille du cluster)

74

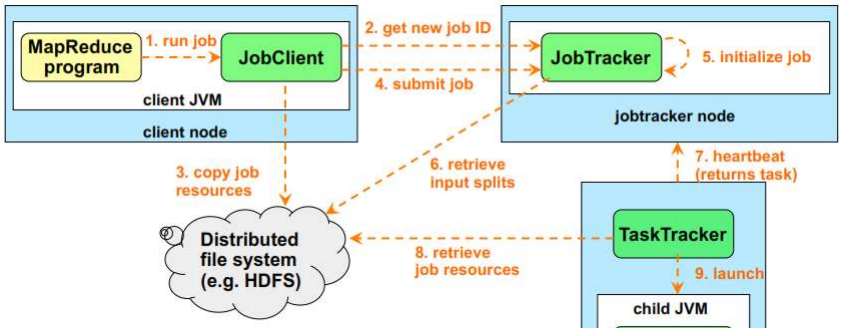
MapReduce: Etapes d'exécution d'un job



7. Les **TaskTrackers** envoient régulièrement des pulsations (Heartbeat) au **JobTracker** pour lui signifier leurs disponibilités à exécuter des jobs. Si le **Jobtracker** possède des jobs dans sa fille d'attente, il confiera leurs tâches **Map** et **Reduce** aux **TaskTrackers**
8. Les **TaskTrackers** doivent obtenir le code à exécuter. Ils le récupèrent à partir du dossier HDFS du job contenant les ressources du job copiées dans l'étape 3.

75

MapReduce: Etapes d'exécution d'un job

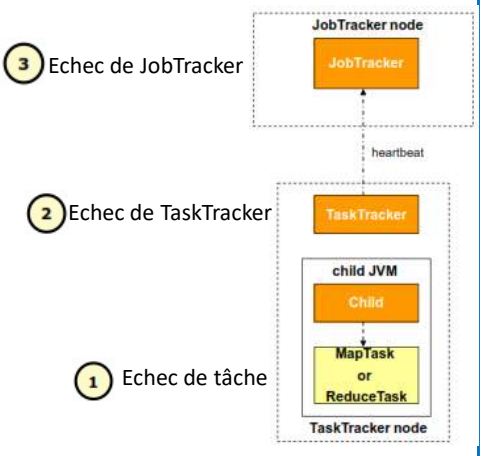


9. Le **TaskTracker** lance une nouvelle machine virtuelle Java (JVM) avec un **processus enfant** qui y est exécuté. Ce processus enfant exécute une tâche **Map** ou **Reduce**. Le résultat d'une tâche **Map** est enregistré sur le disque local du nœud **TaskTracker** (pas dans HDFS).

 CC-BY NC SA

MapReduce: Tolérance aux pannes

- Le moyen principal utilisé par Hadoop pour assurer la tolérance aux pannes consiste à redémarrer des tâches en cas d'échec.
- Le **JobTracker** sait les tâches **Map** et **Reduce** attribuées à chaque **TaskTracker**.
- Si un **TaskTracker** n'envoie pas une pulsation au **JobTracker** pendant une période donnée (par défaut, 10 minute), le JobTracker le considère bloqué.
- En cas d'échec d'un **TaskTracker** pendant l'exécution de ses tâches **Map**, elles seront **toutes** attribuées aux autres **TaskTrackers**.



- En cas d'échec d'un **TaskTracker** pendant l'exécution de ses tâches **Reduce**, les autres **TaskTrackers** ré-exécuteront **seulement** les tâches **Reduce en cours** sur le **TaskTracker** en échec.

 CC-BY NC SA

MapReduce 1: Maitre/Esclave

- Dans l'architecture **MapReduce**, l'exécution de jobs est contrôlée par deux types de processus:
  - Le **JobTracker**, l'unique maître, qui coordonne et attribue toutes les tâches **Map** et **Reduce** exécutées sur le cluster,
  - Un certain nombre de **TaskTrackers**, processus subordonnés, qui exécutent les tâches assignées et rendent compte périodiquement de leur avancement au **JobTracker**.

 CC-BY NC SA

MapReduce 1: Responsabilités du JobTracker

- Dans **MapReduce 1**, le **JobTracker** est chargé de **deux responsabilités** distinctes:

1. **Gestion des ressources et ordonnancement des jobs** de calcul dans le cluster, ce qui implique de gérer la liste des **nœuds actifs**, la liste des **Map** et **Reduce** et des emplacements (**Slots**) disponibles et occupés, ainsi que d'affecter les emplacements disponibles aux jobs et tâches appropriés, en fonction de la politique d'ordonnancement sélectionnée (FIFO, ….).
2. **Coordination de toutes les tâches** en cours d'exécution sur un cluster, ce qui implique de donner aux **TaskTrackers** l'ordre de démarrer des tâches **Map** et **Reduce**, de surveiller l'exécution des tâches, de relancer les tâches ayant échoué, d'exécuter **spéculativement** des tâches lentes, … etc.

 CC-BY NC SA

MapReduce 1 – Limites

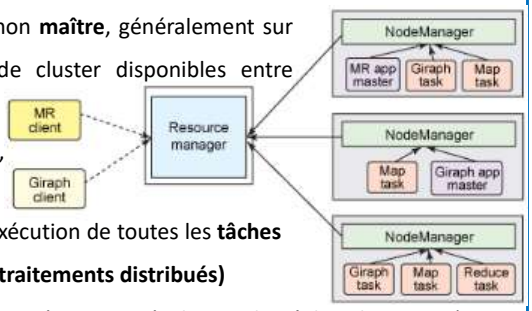
- Les limitations les plus sérieuses de **MapReduce** V1 sont:
  - **L'évolutivité**: Les **responsabilités** du **JobTracker** posaient d'importants problèmes d'évolutivité: **MapReduce1** rencontre des goulots d'étranglement d'évolutivité lorsqu'elle s'approche de 4 000 nœuds et 40 000 tâches, ceci est dû au fait que le **JobTracker** gère à la fois les **jobs** et leurs **tâches**.

    YARN surmonte ces limitations grâce à son architecture de **Ressource Manager/Application Master** séparés: il est conçu pour évoluer jusqu'à 10 000 nœuds et 100 000 tâches.
  - **L'utilisation des ressources**: Les **TaskTrackers** dans **MapReduce 1** sont configurés avec un nombre de **Slots** divisés **séparément** en slots pour les tâches **Map** et d'autres pour les tâches **Reduce**. **Yarn** utilise des **conteneurs** (**container**) pouvant être utilisés pour exécuter n'importe quel type de tâche.
  - N'est pas **Multi-tenant**: pas de prise en charge de job non **MapReduce**.

 CC-BY NC SA

YARN (*Yet Another Resource Negotiator*)

- **MapReduce V1** a subi une refonte complète avec **YARN**, scindant les deux fonctionnalités principales de **JobTracker** (*gestion des ressources et planification / surveillance des jobs*) en démons distincts.
- **Resource Manager (RM)**
  - L'unique **Resource Manager** s'exécute tant que démon **maître**, généralement sur une machine dédiée. Il partage les ressources de cluster disponibles entre diverses applications (jobs) concurrentes.
  - Lorsqu'un utilisateur soumet une **application (job)**, une instance d'un processus léger appelé **ApplicationMaster** est démarrée pour coordonner l'exécution de toutes les **tâches de cette application (selon plusieurs paradigmes de traitements distribués)**



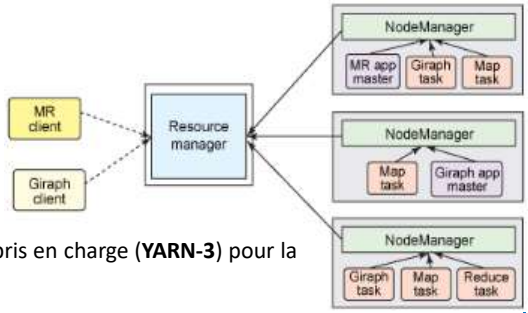
(*surveillance, redémarrage de tâches ayant échoué, exécution spéculative de tâches lentes, ….*) Ces responsabilités étaient précédemment attribuées au **JobTracker** pour **tous les jobs**.

  - L'**ApplicationMaster** et les **tâches** de son application s'exécutent dans des **conteneurs** de ressources contrôlés par les **NodeManagers**

 CC-BY NC SA

YARN (Yet Another Resource Negotiator)

- **NodeManager (NM)**



- Il dispose d'un nombre de **conteneurs** de ressources créés dynamiquement. La capacité d'un conteneur dépend de la quantité de ressources qu'il contient (*mémoire, CPU, disque, E/S réseau*)
- Actuellement, seuls la mémoire et le processeur sont pris en charge (**YARN-3**) pour la la configuration des conteneurs..
- Le **NodeManager** démarre les conteneurs, les surveille et communique leur statuts au **ResourceManager**
- Le nombre de conteneurs sur un nœud dépend de paramètres de la configuration et le nombre total de ressources du nœud (*nombre total de processeurs et taille mémoire*)
- **ApplicationMaster** peut **exécuter** **n'importe quel type de tâche** dans un **conteneur**. Par exemple, **ApplicationMaster MapReduce** demande à un conteneur de lancer une tâche **Map** ou **Reduce**, tandis qu'un **ApplicationMaster Giraph** demande à un conteneur d'exécuter une tâche **Giraph**.

82

 CC-BY NC SA

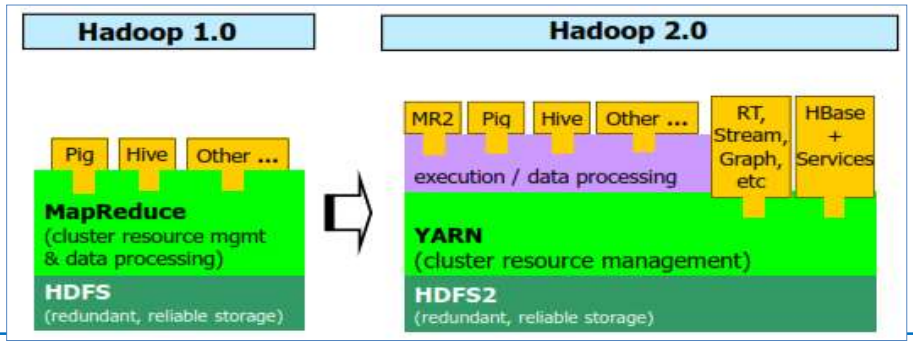
YARN (Yet Another Resource Negotiator)

Hadoop v1 Vers Hadoop v2

MapReduce 1	YARN
Jobtracker	Resource manager, application master,
Tasktracker	Node manager
Slot	Container

Système à usage unique
Traitement par lots

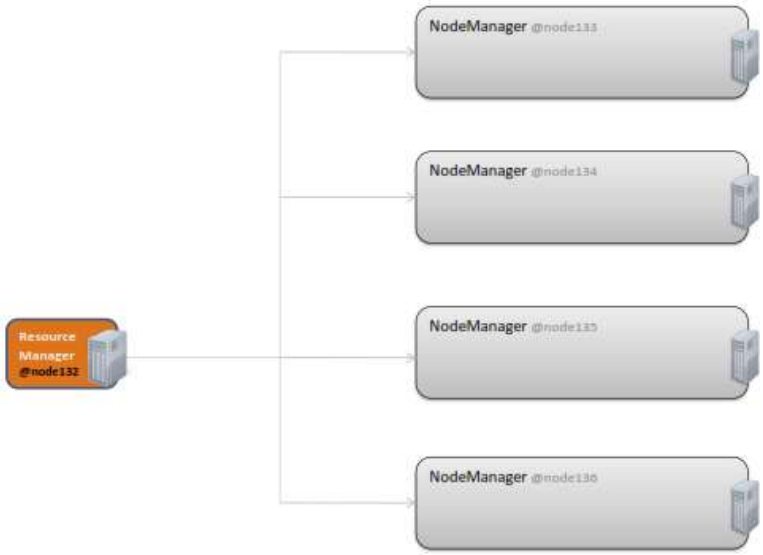
Plateforme à usages multiples
Traitement par lots, interactif, en ligne, streaming



83

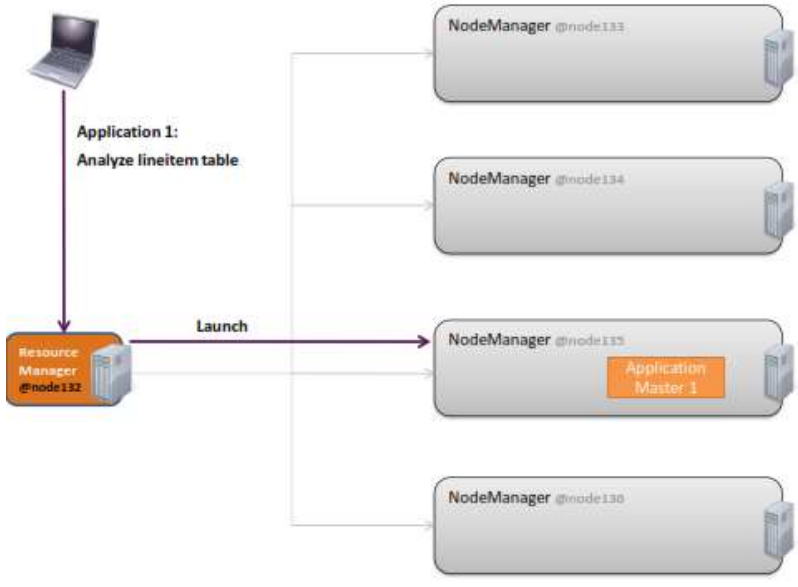
 CC-BY NC SA

YARN: Exécuter une application dans Yarn (1 / 7)



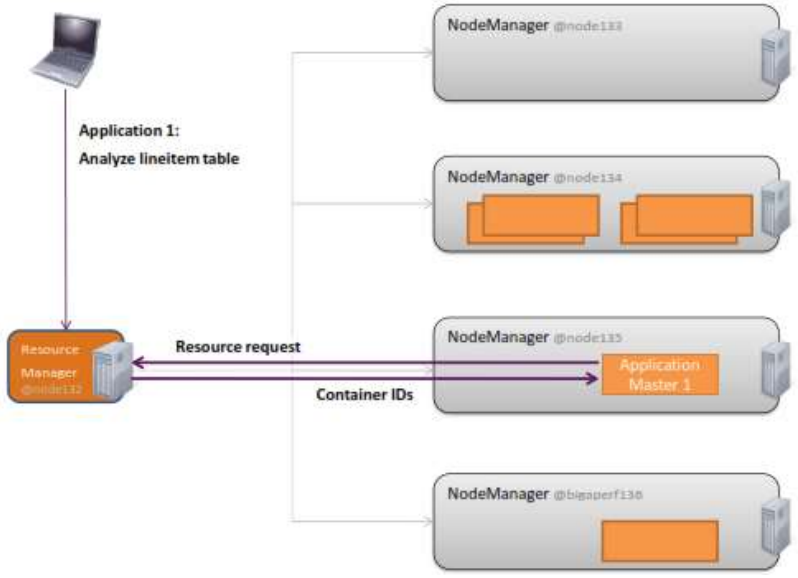
 CC-BY NC SA

YARN: Exécuter une application dans Yarn (2 / 7)



 CC-BY NC SA

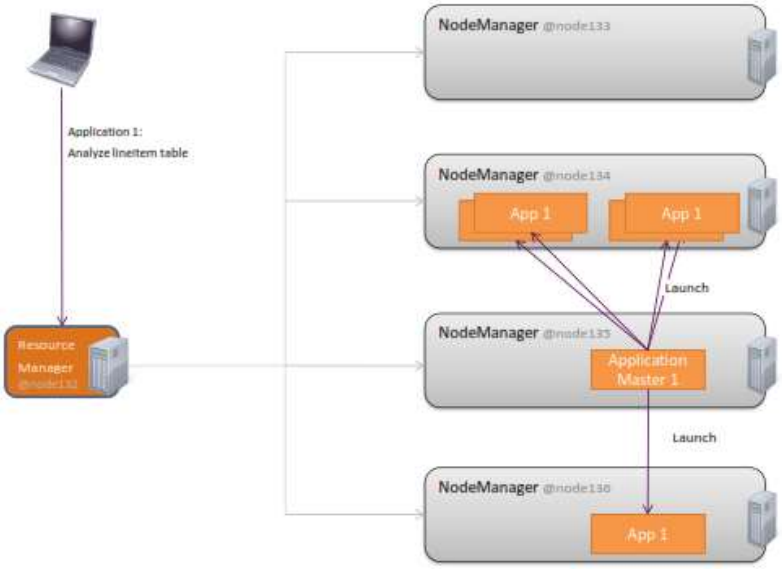
YARN: Exécuter une application dans Yarn (3 / 7)



86

 CC-BY NC SA

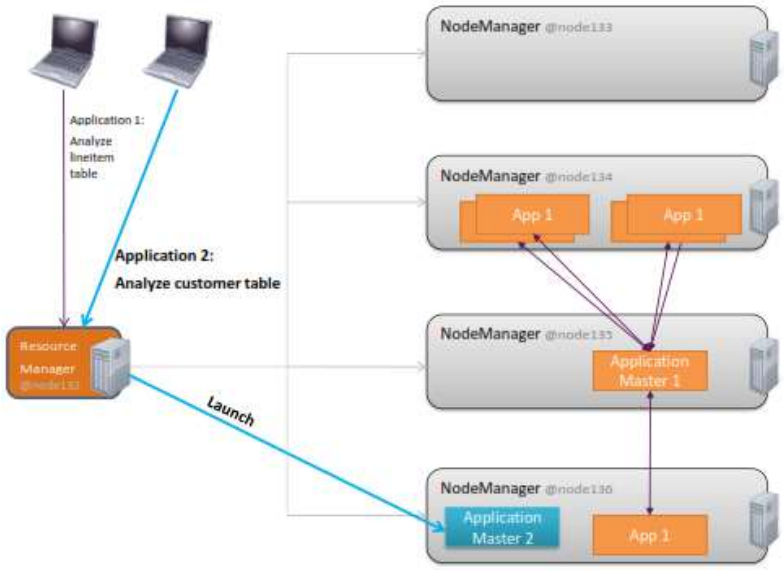
YARN: Exécuter une application dans Yarn (4 / 7)



87

 CC-BY NC SA

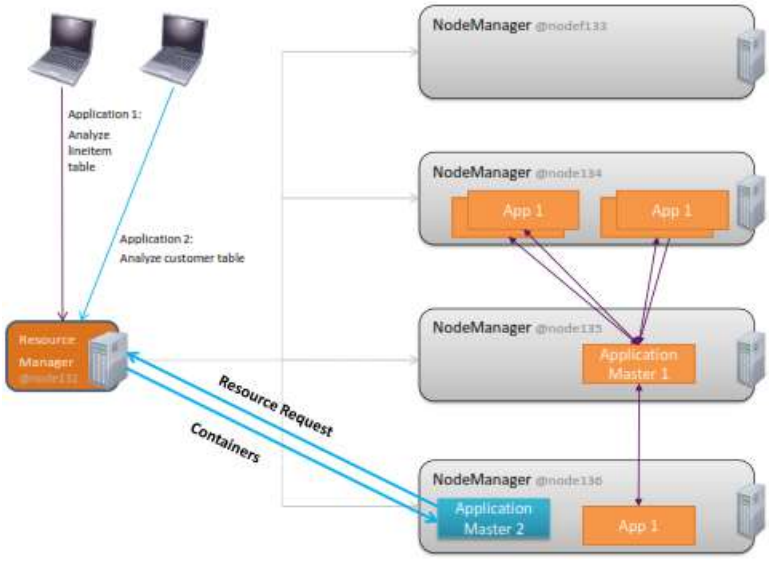
YARN: Exécuter une application dans Yarn (5 / 7)



88

 CC-BY NC SA

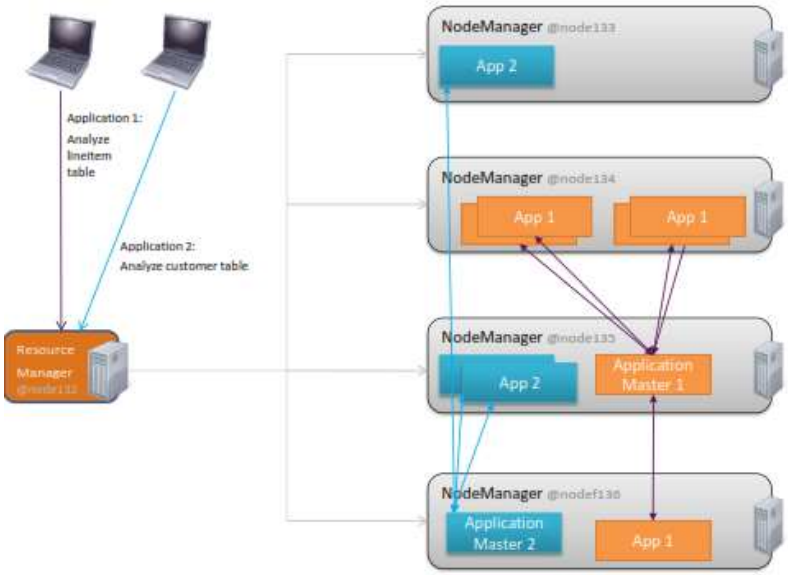
YARN: Exécuter une application dans Yarn (6 / 7)



89

 CC-BY NC SA

YARN: Exécuter une application dans Yarn (7 / 7)



Atelier 2 *(1/5)*

1. Copier le dossier **Tp_MapRed** dans le dossier local **/home/cloudera** de votre VM.
2. Créer les dossiers **HDFS** suivants: **/user/cloudera/wordcount/input**
3. Charger le fichier texte **maman.txt** dans HDFS: **/user/cloudera/wordcount/input**
4. **Exécuter le job MapReduce comme suit:**

$ cd **/home/cloudera/Tp_MapRed/WordCount**

Exécution de **wordcount.jar** dont la classe principale est **WordCount** du package **org.myorg**

$ **hadoop jar** ***wordcount.jar*** org.myorg.***WordCount***
/user/cloudera/wordcount/input **/user/cloudera/wordcount/output**

Les arguments (args[0] et args[1]) de la fonction main() dans la classe principale.

Afficher le contenu de tous les fichiers du dossier HDFS **/user/cloudera/wordcount/output/** qui contient les résultats produits par les Reducers.

$ **hdfs dfs** -cat /user/cloudera/wordcount/output/*

Atelier 2 *(2/5)*

1. Faire une copie de **WordCount.java** dans **WordTotal.java**
2. Faire ces modifications dans **WordTotal.java :**
   - Remplacer **WordCount** par **WordTotal**
   - Remplacer **wordcount** par **wordtotal**
3. **Modifier** la méthode **Map** dans **WordTotal.java** pour ***calculer le nombre total de mots.***

   N.B: Le résultat finale doit être **un seul pair** <*clé, valeur*>, par exemple <Nombre_mots, 366)

3. Compiler, Générer et Exécuter votre job:

$ rm -rf build **build** est le dossier pour stocker le ByteCode généré lors de la compilation

$ mkdir build

$ **javac -cp** /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* ***WordTotal.java*** -d **build** -Xlint

$ **jar -cvf wordtotal.jar -C build/ .**

 CC-BY NC SA

Atelier 2 *(3/5)*

$ **hdfs dfs** -rm -r -f /user/cloudera/wordcount/output

$ **hadoop jar** ***wordtotal.jar*** org.myorg.***WordTotal*** **/user/cloudera/wordcount/input**

/user/cloudera/wordcount/output

$ **hdfs dfs** -cat /user/cloudera/wordcount/output/*

 CC-BY NC SA

Atelier 2 *(4/5)*

1. Charger le fichier **vol.csv** dans le dossier **HDFS** à créer: ***/user/cloudera/data_vol/input***

   La structure de **vol.csv** est la suivante:

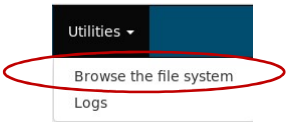
   année;mois;jour:num_vol;depart;arrivée;distance

2. Lister le contenu du dossier HDFS: ***/user/cloudera/data_vol/input***

3. Utiliser la commande **hdfs fsck** pour avoir un rapport détaillé sur les blocs du fichier **/user/cloudera/data_vol/input /vol1.csv**

   $ hdfs fsck /user/cloudera/data_vol/input/vol.csv -blocks

4. Utiliser l'url **http://localhost:50070** pour parcourir l'arborescence HDFS via le lien Utilities



 CC-BY NC SA

Atelier 2 *(5/5)*

6. - Enregistrer **WordCount.java** sous le nom **Vol.java**

   - Modifier **Vol.java** afin d'avoir le **nombre de vols en partance de chaque aéroport**.

```
public void map (LongWritable key , Text value, Context context)
  throws IOException, InterruptedException {
        String line = value.toString();      String [] word = line.split(";");
        context.writttte(…….. , …… )
                                    }
```

   - Compiler et exécuter le programme

7. Modifier une copie de votre programme afin d'avoir la **distance maximale**.

8. Modifier une copie de votre programme afin d'avoir le **nombre de vols** pour chaque **pair d'aéroports**, sans distinction entre départ et arrivée, Exemple: IND-BWI 24

9. Modifier le programme de la question 6 afin d'avoir le nombre de vols en partance et en arrivé pour chaque aéroport. Exemple de résultat : IND Départ 136

   IND Arrivée 8

 CC-BY NC SA