Écosystème Hadoop

Hmida HMIDA

10 octobre 2022

Table des matières

Présentation

1

Objectifs

Objectifs

- Retracer l'évolution chronologique de Hadoop
- Énumérer les composants et les caractéristiques de Hadoop
- Découvrir le fonctionnement du système HDFS
- Comprendre le principe du modèle Map/Reduce
- Écrire des programmes Map/Reduce

Historique

2002 : Projet Nutch par Doug Cutting et Mike Cafarella
2003 : Publication par Google de Whitepaper sur GFS
2004 : Publication par Google de Whitepaper sur Map/Reduce
2004 : Implémentation de NDFS et Map/Reduce dans Nutch
2006 : Hadoop comme sous projet de Nutch
- Doug Cutting chez Yahoo
- Nom de la peluche de son enfant

2007 : Utilisation de Hadoop par Yahoo sur un cluster de 1000 noeuds
2008 : Hadoop comme un projet Apache http://hadoop.apache.org
2011 : Version 1.0 de Hadoop
2012 : version 2.0 ➔ Introduction de YARN
2017 : Version 3.0 ➔ Erasure Coding, Haute disponibilité

Système distribué

Définition :
- Un système distribué est une collection de processus ou d'ordinateurs indépendants et coopératifs qui apparaissent à l'utilisateur comme un seul et unique système cohérent

Objectifs
- Hétérogénéité : architecture, protocoles, .. différents
- « Scalabilité » : Rester efficace en cas d'augmentation des ressources et des utilisateurs :
  - ▶ Mise à l'échelle verticale
  - ▶ Mise à l'échelle horizontale
- Tolérance aux pannes : disponibilité même en conditions dégradées
- Concurrence : accès aux ressources partagées
- Transparence pour l'utilisateur : Système vu et utilisé comme une seule entité

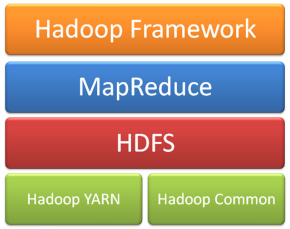
Pourquoi Hadoop?

Projet Open Source (en Java)
- License
- Dynamique
- Communauté

Apporte des solutions aux défis de la gestion des données massives
- Stockage : Systèmes HDFS
- Traitement : Map/Reduce et YARN

Avantages :
- Scalabilité linéaire (taille + temps)
- Disponibilité
- Les traitements se déplacent vers les donnéés
- Traitement séquentiel (non aléatoire)
- Modèle de traitement simple

Architecture (voir figure)



Comment obtenir Hadoop?

Installer depuis les binaires sur hadoop.apache.org
- Sur Linux mais aussi Windows et Mac
- Difficulté d'administration et intégration avec les autres outils

Distributions Hadoop
- Environnement pré-installé avec outils d'administration
- Packagé en machine virtuelle ou image Docker
- Fournisseurs :
  - ▶ Cloudera : (après fusion avec Hortonworks en 2018) : HDP et HDF
  - ▶ HP Enterprise : hérite de MapR rebaptisée HPE Ezmeral Data Fabric
  - ▶ IBM : IBM Open Platform ou IBM Insights

Équipement dédié :
- Matériel optimisé pour Hadoop comme : Dell, EMC, Teradata Appliance for Hadoop, HP, Oracle, ...

Service Cloud : PaaS (Platform as a Service) tel que Amazon EMR, Microsoft HDInsight, Google Cloud Platform, Qubole, IBM BigInsights, ...

Écosystème Hadoop

Hadoop
- HDFS
- Map/Reduce
- YARN

Outils complémentaires
- Ingestion de données
- Traitement et Interrogation
- Coordination et orchestration
- Surveillance et monitoring
- ...

Écosystème Hadoop

Quelques outils :
- **Spark :** Moteur de traitement en mémoire distribuée et performant avec support de SQL, Streaming et Apprentissage automatique et des graphes
- **Hive :** Datawarehouse au dessus de Hadoop avec syntaxe SQL
- **Hbase :** BD non relationnelle orientée colonne au dessus de HDFS
- **Ambari :** gestion et déploiement de cluster Hadoop
- **Sqoop :** Transfert de données depuis sources externes
- **Kafka :** Traitement de flux de messages en producteur/consommateur
- **Zookeeper :** service centralisé pour la gestion de configuration
- **Nifi :** Automatisation des flux de données entre systèmes
- **Storm :** Traitement de flux en temps réel
- **Flume :** Collecte, agrégation et transfert de logs
- **Flink :** Traitement distribué de flux de données avec état (stateful)

Hadoop –implantations

Yahoo (2010) :
- 70 millions de fichiers, 80 millions de blocs
- 15 PB
- +4000 noeuds, 24000 clients
- 50 GB métadonnées en RAM

Facebook (2010) :
- 55 millions de fichiers, 80 millions de blocs
- 21 PB
- 2000 noeuds, 30000 clients
- 108 GB métadonnées en RAM

HDFS

2

Introduction

Hadoop Distributed File System
- Système de fichiers
- Distribué
- Autres DFS : CEPH, GlusterFS, OpenIO, BeeGFS, ...

Gérer des fichiers volumineux

Déployé sur un cluster de noeuds avec des disques : Commodity hardware

Indépendant de l'OS

Unité de stockage : bloc de taille configurable (128 MO par défaut)

Réplication des blocs sur des nœuds différents (3 par défaut)

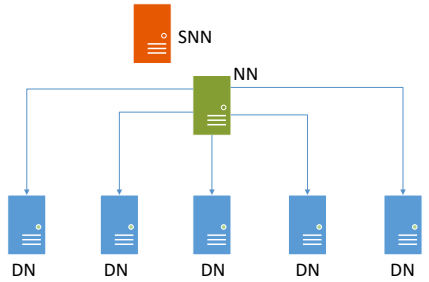
Composants HDFS

Architecture Maître/Esclave

NameNode (NN)
- Gère les métadonnées sur les fichiers, répertoires et blocs
- Métadonnées chargées en RAM

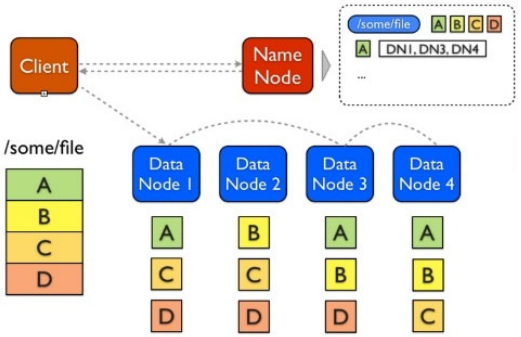
DataNode (DN)
- Contient les blocs de données
- Opérations L/E
- Envoie des heartbeats et rapports de blocs au NN

Secondary NameNode (SNN)
- NN est un point critique : SPOF
- Surveille le NN et copie les métadonnées
- Remplace le NN en cas de panne



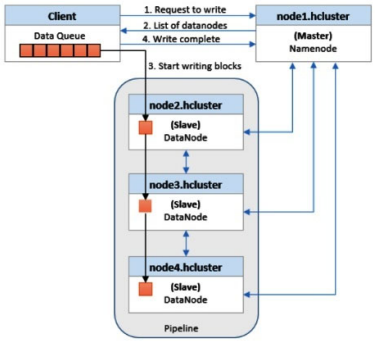
Cluster HDFS

Exemple



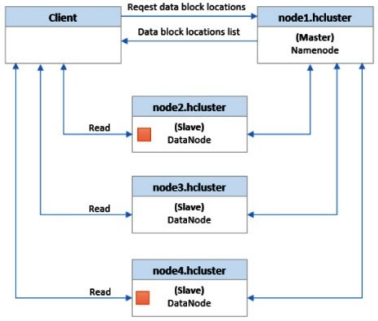
Écriture

1. Le client HDFS envoie une requête d'écriture au NN
2. Le NN crée le fichier dans son namespace, calcule une topologie de blocs et répond par une liste de DNs
3. Le client prépare les blocs et envoie le flux de données au premier DN qui va relayer les données aux autres DNs
4. Les DNs échangent des accusés pour signaler la réussite/échec de l'écriture.



Lecture

1. Le client HDFS envoie une requête de lecture au NN
2. Le NN répond par une liste de DNs hébergeant le bloc demandé triée selon la proximité au client
3. Le client se connecte au premier DN pour lire le bloc
4. En cas d'erreur passe au DN suivant



Stratégie de réplication

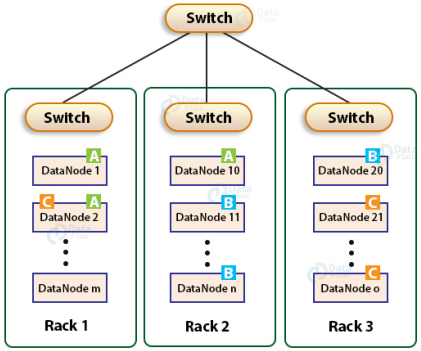
1er bloc sur DN local

2ème bloc sur un autre DN du même Rack

3ème bloc sur DN d'un autre Rack

Pour le reste, respecter :

- Pas de copies sur le même DN.
- Pas plus de 2 copies sur le même Rack.
- Nombre de Racks utilisés < nombre de copies.



Autres aspects

Permissions : rwx

Erasure Coding

- Inspiré de la technologie RAID
- Remplace la réplication intégrale de blocs
- Réduire le volume de données de +50%
- Pour 2 blocs data calculer 1 bloc de parité

Fédération HDFS

- Plusieurs NN
- Plusieurs namespaces sur le même cluster (NN1 : /user, NN2:/data)

Interfaces

API Java

API C : libhdfs

Web : sur https://namenode:9870

CLI :

```
hadoop fs –commande -option arguments
hdfs dfs –commande -option arguments
```

Commandes :

- ls : afficher le contenu du dossier
- put : copier du système local vers HDFS
- get : copier de HDFS vers le système local
- cp : copie de fichiers/dossiers HDFS
- mv : déplacement de fichiers/dossiers HDFS
- touch : créer un fichier vide
- cat : afficher le contenu d'un fichier
- mkdir : créer un dossier
- rm : supprimer fichiers
- rmdir : supprimer un dossier vide

Limites

N'est pas adapté pour les systèmes à faible latence

Lent avec les fichiers de taille réduite

Convient pour le mode ajout et non modification aléatoire

Tout est en mémoire : Problème de scalabilité

MapReduce

3

Présentation

Calcul distribué/parallèle

Parallèle

- Threads d'exécution simultanés
- Mémoire partagée ➔ synchronisation
- Passage à l'échelle vertical
- SPOF (Single Point Of Failure) : point vulnérable

Distribué

- Nœuds autonomes
- Collaboration par échange de messages
- Passage à l'échelle horizontal
- Tolérance aux pannes

Modèle de programmation algorithmique

Diviser pour régner (divide and conquer)

Décomposer le problèmes en tâches indépendantes et « parallélisables »

Combinaison de 2 fonctions **Map** et **Reduce**

Fonctions Map et Reduce

En programmation fonctionnelle

map() : Appliquer une fonction sur tous les éléments d'une liste

```
>>> list(map(lambda x:x**2, [2,5,8,10]))
[4, 25, 64, 100]
```

reduce() : Applique une fonction d'agrégation sur une liste

```
>>> from functools import reduce
>>> reduce(lambda x,y:x+y, [1,2,3,4])
10
```

Étapes MapReduce

Split : découper
- Fragmenter les données en des lots plus petits

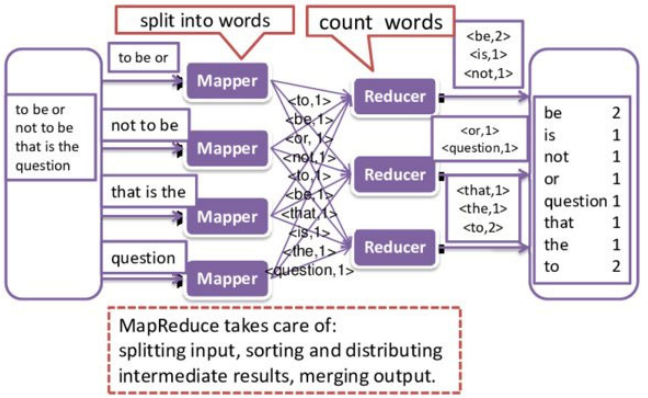
Map
- Appliquer sur chaque lot la fonction spécifiée par le problème
- Transforme le lot en une liste de paires (clé, valeur)
- Résultats intermédiaires

Shuffle & sort : mélanger et trier
- Les résultats intermédiaire sont regroupés et triés par clé

Reduce
- Appliquer la fonction reduce et agréger les valeurs relatives à une clé en une seule valeur

Exemple : Wordcount

Problème : compter le nombre d'occurrences des mots dans un fichier texte



Programmation MapReduce

Le rôle du développeur :
- Choisir une manière de découper les données afin que l'opération MAP soit parallélisable.
- Choisir la clé à utiliser pour le problème ciblé.
- Écrire le code de la fonction pour l'opération MAP.
- Écrire le code de la fonction pour l'opération REDUCE.

Framework Hadoop
- Partitionnement des données
- Tolérance aux fautes et reprise après pannes
- Ordonnancement
- Shuffle & sort
- Allocation des ressources

Programme Wordcount

Comment découper : découper un texte en mots

Clé / Valeur : Résultat = liste de paires (mot, nombre d'occurrences)

Map

```
fonction Map(clé, valeur):
// clé : nom_fichier, valeur : contenu (bloc, ligne, ...)
Pour chaque mot dans valeur:
    Émettre(mot, 1)
```

Reduce

```
fonction Reduce(clé, valeurs):
    //clé = mot, valeurs = liste de 1 (nombre de 1 = nombre d'occurrences)
    occurrences = 0
    Pour chaque v dans valeurs:
        Occurrences = occurrences + v
        Émettre (clé, occurrences)
```

Worcount en Java –Mapper


```java
import ...;
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);

    private Text word = new Text();
    @Override

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {

            word.set(tokenizer.nextToken());

            context.write(word, one);

        }

    }
    public void run(Context context) throws IOException, InterruptedException {

        setup(context);

        while (context.nextKeyValue()) {

            map(context.getCurrentKey(), context.getCurrentValue(), context);

        }

        cleanup(context);

    }
}
```


Wordcount en java –Reducer


```java
import ...;


public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {


    private IntWritable totalWordCount = new IntWritable();
    @Override

    public void reduce(final Text key, final Iterable<IntWritable> values,

            final Context context) throws IOException, InterruptedException {
        int sum = 0;

        Iterator<IntWritable> iterator = values.iterator();
        while (iterator.hasNext()) {

            sum += iterator.next().get();

        }
        totalWordCount.set(sum);

        // context.write(key, new IntWritable(sum));

        context.write(key, totalWordCount);

    }
}
```


Wordcount en java –Driver


```java
import ...;
public class WordCountDriver extends Configured implements Tool {

    public int run(String[] args) throws Exception {

        if (args.length != 2) {

            System.out.println("Usage: [input] [output]");

            System.exit(-1);

        }

        // Creation d'un job en lui fournissant la configuration et une description textuelle de la tache

        Job job = Job.getInstance(getConf());

        job.setJobName("wordcount");
        // On precise les classes MyProgram, Map et Reduce

        job.setJarByClass(WordCountDriver.class);

        job.setMapperClass(WordCountMapper.class);

        job.setReducerClass(WordCountReducer.class);
        // Definition des types clé/valeur de notre problème

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);

        Path inputFilePath = new Path(args[0]);

        Path outputFilePath = new Path(args[1]);

        // On accepte une entree recursive

        FileInputFormat.setInputDirRecursive(job, true);

        FileInputFormat.addInputPath(job, inputFilePath);

        FileOutputFormat.setOutputPath(job, outputFilePath);
        FileSystem fs = FileSystem.newInstance(getConf());

        if (fs.exists(outputFilePath)) {

            fs.delete(outputFilePath, true);

        }

        return job.waitForCompletion(true) ? 0: 1;

    }

    public static void main(String[] args) throws Exception
    {

        WordCountDriver wordcountDriver = new WordCountDriver();

        int res = ToolRunner.run(wordcountDriver, args);

        System.exit(res);

    }

}
```


Wordcount en Python

Bibliothèque hadoop-streaming.jar

Les flux standards pour les E/S

mapper.py

```python
import sys
for line in sys.stdin:
    # recupérer les mots
    words = line.split()
    # operation map, pour chaque mot, generer la paire (mot, 1)
    for word in words:
        print("%s\t%d" % (word, 1))
```

Wordcount en Python

reducer.py

```
import sys

total = 0

lastword = None
for line in sys.stdin:

    line = line.strip()
    # recuperer la cle et la valeur et conversion de la valeur
en int

    word, count = line.split()

    count = int(count)
    # passage au mot suivant (plusieurs cles possibles pour une
 même exécution de programme)

    if lastword is None:

        lastword = word

    if word == lastword:

        total += count

    else:

        print("%s\t%d occurences" % (lastword, total))

        total = count

        lastword = word

if lastword is not None:

    print("%s\t%d occurences" % (lastword, total))
```

Exécution

```
hadoop jar hadoop-streaming.jar -input shakespeare.txt -
output /results -mapper mapper.py -reducer reducer.py
```

Résultat

```
hadoop fs -cat /results/*
```


Wordcount en Python avec MRJOB

Wordount.py

```
from mrjob.step import MRStep
from mrjob.job import MRJob
class WordCount(MRJob):
    def steps(self):
        return [MRStep(mapper=self.mapper_get_words,
                reducer=self.reducer_count_words)]
    def mapper_get_words(self, _, line):
        for word in line.split():
            yield word, 1
    def reducer_count_words(self, key, values):
        yield key, sum(values)
if __name__ == '__main__':
    WordCount.run()
```

Exécution

```
python wordcount.py -r hadoop hdfs://localhost:9000/user/hadoop/shakespeare.txt
```

4 YARN

Hadoop V1

- Pas de YARN
- Unique schéma accepté Map/Reduce
- Application MapReduce = Job MapReduce
- Modèle Master/Slave : Job Tracker/Task Tracker
  - Diviser le job sur plusieurs tâches appelées mappers et reducers
  - Chaque tâche est exécutée sur un nœud du cluster
  - Chaque nœud a un certain nombre de slots prédéfinis: Map Slots/Reduce Slots
- Un slot est une unité d'exécution qui représente la capacité du task tracker à exécuter une tâche (map ou reduce) individuellement, à un moment donné
- Le Job Tracker se charge à la fois:
  - D'allouer les ressources (mémoire, CPU…) aux différentes tâches
  - De coordonner l'exécution des jobs MapReduce
  - De réserver et ordonnancer les slots, et de gérer les fautes en réallouant les slots au besoin

Hadoop V2

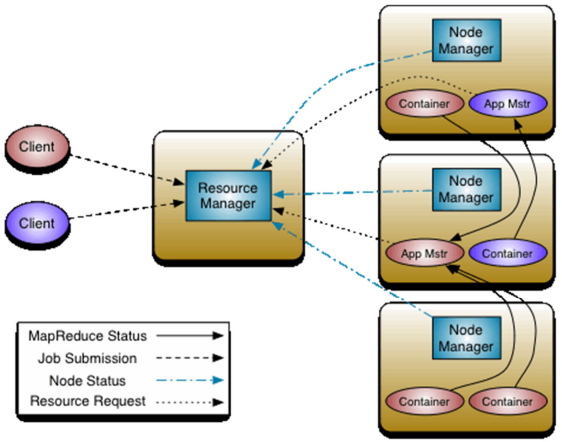
Découplage de la gestion des ressources de la gestion des tâches

Pas de slots mais des ressouces (RAM, Cores CPU) allouées à la demande

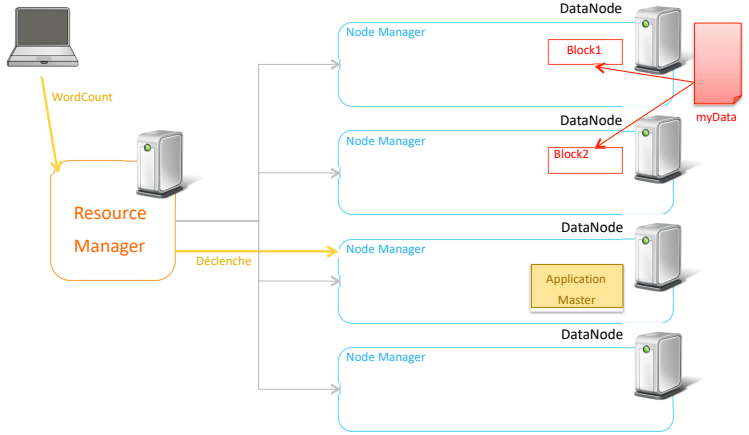
YARN : Yet Another Resource Negotiator

- Resource Manager (RM)
  - ▶ Master
  - ▶ Arbitrage des ressources entre plusieurs applications
- Node Manager (NM)
  - ▶ Slave
  - ▶ Création de containers et allocation des ressources
- Application Master (AM)
  - ▶ 1 par application
  - ▶ Sur 1 container
  - ▶ Demande des containers pour les tâches de l'application

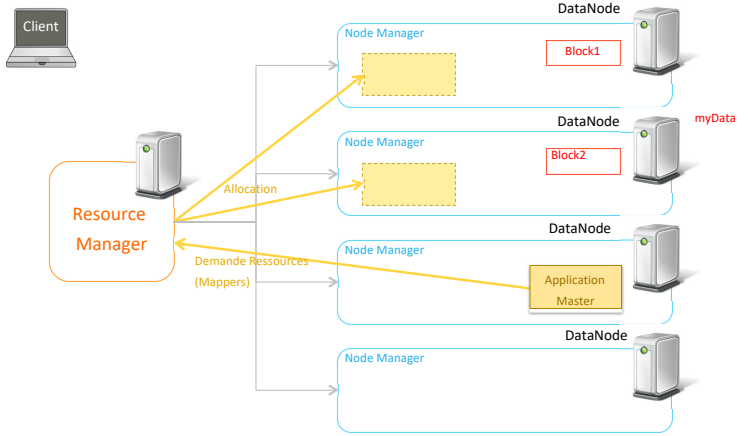
Application sous YARN



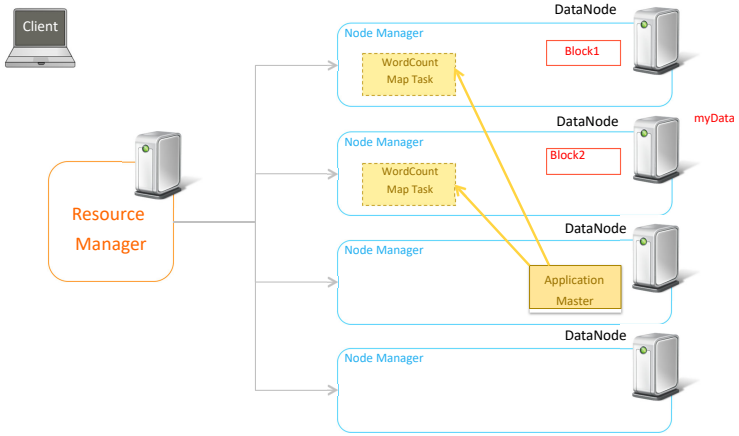
Job MR avec YARN (1)



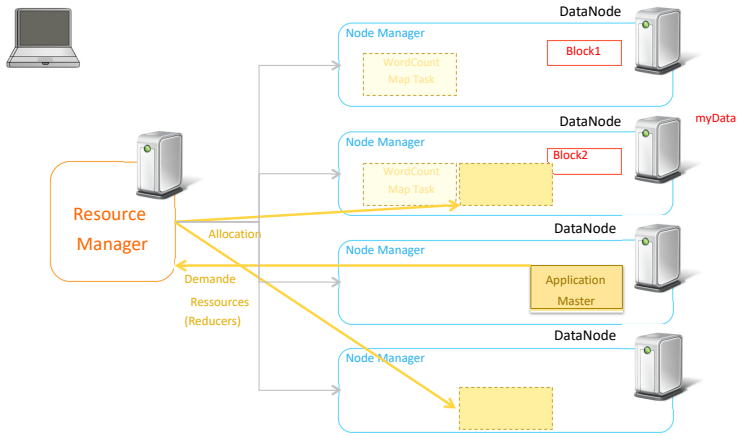
Job MR avec YARN (2)



Job MR avec YARN (3)



Job MR avec YARN (4)



5 Exemples MR

Filtrage avec MapReduce

Filtrer : sélection selon un critère

Exemple 1 : Garder les mots de taille supérieure à 10

Ignorer les mots <10

```
from mrjob.step import MRStep
from mrjob.job import MRJob
class Word10(MRJob):
    def steps(self):
        return [MRStep(mapper=self.mapper_filter, reducer=self.reducer_print_words)]
    def mapper_filter(self, _, line):
        for word in line.split():
            if len(word)>10:
                yield word, 1
    def reducer_print_words(self, key, values):
        yield key, len(key)
if __name__ == '__main__':
    Word10.run()
```

Filtrage avec MapReduce

Exemple 2 : Afficher les 10 mots les plus longs

Mapper : Une liste locale

Reducer : La liste globale

```
from mrjob.step import MRStep
from mrjob.job import MRJob
class Top10(MRJob):
    def steps(self):
        return [MRStep(mapper=self.mapper_top10, reducer=self.reducer_top10)]
    def mapper_top10(self, _, line):
        word_list=line.split()
        word_list.sort(key=lambda w:len(w), reverse=True)
        yield _, word_list[:10]
    def reducer_top10(self, key, values):
        word_list = []
        for wl in values:
            for w in wl:
                word_list.append(w)
        word_list.sort(key=lambda w:len(w), reverse=True)
        for i in range(10):
            yield i+1,word_list[i]
if __name__ == '__main__':
    Top10.run()
```


Agrégation avec MapReduce

Wordcount est un exemple d'agrégation

Autres :

- Min, Max
- Moyenne
- Premier, dernier
- ...

Clé/valeur :

En SQL :

```
SELECT fonction_aggrégation(valeur)
FROM ...
GROUP BY (clé)
```

Combiner :

- Exécution du Reducer localement sur chaque Mapper

Agrégation avec MapReduce

Exemple : Salaire moyen par département

Format CSV : id_emp, nom, salaire, dep

```
from mrjob.step import MRStep
from mrjob.job import MRJob
class Avg(MRJob):
    def steps(self):
        return [MRStep(mapper=self.mapper_salaire, reducer=self.reducer_avg)]
    def mapper_salaire(self, _, line):
        employe=line.split(',')
        yield employe[3], (employe[2],1)
    def reducer_avg(self, key, values):
        somme = N = 0
        for v in values:
            somme += int(v[0])
            N+=int(v[1])
        yield key,somme/N
if __name__ == '__main__':
    Avg.run()
```


Jointure avec MapReduce

Ajouter une colonne indiquant le nom de la table pour chaque enregistrement

Fusionner les 2 tables

Mapper :

- Clé : attribut de jointure
- Valeur : enregistrement

Reducer :

- Reçoit la liste des enregistrements par clé de jointure
- Effectuer la jointure en choisissant un enregistrement de chaque table

Exemple de Jointure (1)

Films

ID_realisateur	Titre
123	Pulp Fiction
4567	Le pianiste
234	La leçon de piano
123	Reservoir dogs
..	...

Réalisateurs

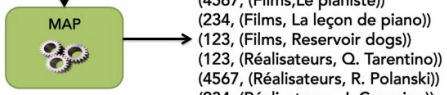
ID_réalisateur	Nom
123	Quentin Tarentino
4567	Roman Polanski
234	Jane Campion
...	...

Films	123	Pulp Fiction
Films	4567	Le pianiste
Films	234	La leçon de piano
Films	123	Reservoir dogs
Réalisateurs	123	Q. Tarentino
Réalisateurs	4567	R. Polanski
Réalisateurs	234	J. Campion
...	...	...

Exemple de Jointure (2)

Mapper

Films	123	Pulp Fiction
Films	4567	Le pianiste
Films	234	La leçon de piano
Films	123	Reservoir dogs
Réalisateurs	123	Q. Tarentino
Réalisateurs	4567	R. Polanski
Réalisateurs	234	J. Campion
...	...	...

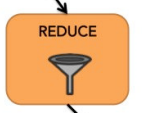


(123, (Films, Pulp Fiction))
(4567, (Films,Le pianiste))
(234, (Films, La leçon de piano))
(123, (Films, Reservoir dogs))
(123, (Réalisateurs, Q. Tarentino))
(4567, (Réalisateurs, R. Polanski))
(234, (Réalisateurs, J. Campion))

Exemple de Jointure (3)

Reducer

(123, [(Films, Pulp Fiction), (Films, Reservoir dogs), (Réalisateurs, Q. Tarentino)]



ID_réalisateur	Nom	Titre Films
123	Quentin Tarentino	Pulp Fiction
123	Quentin Tarentino	Reservoir dogs