

Fonctionnalités



<http://hadoop.apache.org>

- Data d'apparition: 2006, créateur: Doug Cutting
- Hadoop: nom de la peluche d'éléphant jaune du fils de Cutting
- Stockage des données: HDFS
- Traitement distribué de données: map & reduce

1

Exploration de Hadoop



Image issue de : <http://developer.yahoo.com/blogs/ydn/posts/2007/07/yahoo-hadoop/>

Etagère de serveurs (Server rack)

3

Exploration de Hadoop

- Hadoop est écrit en java et est conçu pour traiter en **distribué** une énorme quantité de données structurées et non structurées (terabytes, petabytes)
- Les données sont enregistrées sur des serveurs dans un centre de données, dans des étagères (racks) de serveurs.

2

Exploration de Hadoop

- **Hadoop** stocke et traite les données comme un grappe Hadoop (**Hadoop cluster** - ensemble de noeuds/de serveurs).
- Les **serveurs** peuvent être ajoutés ou retirés dynamiquement (d'une façon logique) du grappe car Hadoop est conçu pour être **autonome**.

4

Hadoop est capable de détecter les changements, incluant les échecs, d'être ajusté à ces changements et continue à fonctionner sans interruption.

- Tolérant aux pannes
- Evolutif (scalable)

Historique de Hadoop

- 2006 Hadoop: Dug Cutting (yahoo) reprendre les concepts présentés par google pour résoudre les problèmes rencontrés depuis le projet **Apache Nutch**. et crée une nouvelle version améliorée qu'il appelle Hadoop, le nom de l'éléphant en peluche de son fils



Historique de Hadoop

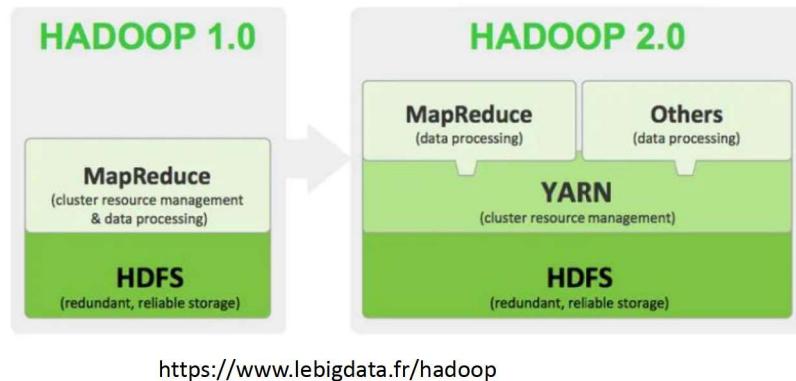
 2002 Development d'un moteur de recherche distribué open source Nutch par Doug Cutting et Mike Cafarella (étudiant). Le calcul se fait sur quelques machines avec quelques limites et problèmes.

- 2003-2004 GFS et Map and reduce: GFS système de fichiers distribués et calcul distribué map-reduce développé par google
- 2004 plateforme pour Nutch: en se basant sur GFS et map-reduce, Dug Cutting développe une plateforme pour Nutch

Historique d'Hadoop

- 2008 Exploitation de Hadoop: projet indépendant de la fondation Apache.
- Yahoo utilise Hadoop pour sa page d'accueil (publicités ciblées, contenus adaptés, le « top recherche » des utilisateurs...). Le grappe (cluster) Yahoo comporte 42000 machines

Composants principaux de Hadoop



9

Composants principaux de Hadoop

Le système de fichiers distribué d'Hadoop (HDFS)

- HDFS reprend de nombreux concepts proposés par des systèmes de fichiers classiques comme ext2 pour Linux ou FAT pour Windows
- HDFS divise les fichiers larges en des petits morceaux appelés *blocs*.
- HDFS est un système distribué.
 - Sur un système classique, la taille du disque est généralement considérée comme la limite globale d'utilisation.

11

- **Hadoop Distributed File System (HDFS):** le Système de fichiers **distribué** Hadoop permet de stocker et de récupérer des fichiers en un temps performant.
- **Moteur MapReduce :** une haute performance d'implémentation de traitement des données parallèle/distribuée de l'algorithme MapReduce.

10

Composants principaux de Hadoop

Le système de fichiers distribué d'Hadoop (HDFS)

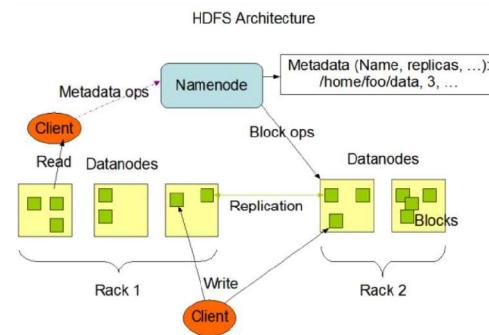
- Dans un système distribué comme HDFS, chaque nœud d'un cluster correspond à un sous-ensemble du volume global de données du cluster.
 - Pour augmenter ce volume global, il suffira d'ajouter de nouveaux nœuds.
- HDFS utilise des tailles de blocs largement supérieures à ceux des systèmes classiques.

12

- Les tailles de Blocs est par défaut 64 MB. Elle peut être configurables à 128 megabytes (MB) ou 256MB. Par exemple un fichier de 1GB consomme huit blocs de 128MB à son stockage.
- La taille pour un système classique est de 4 KO. Les tailles plus grandes permettent d'accélérer l'accès à un bloc.
- Pour éviter les pertes de données, HDFS réplique les blocs sur plusieurs noeuds

13

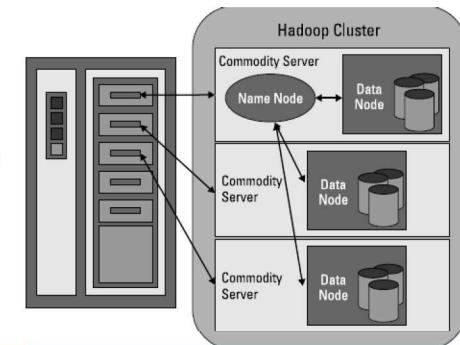
- Les blocs sont stockés et répliqués. Le facteur de duplication est configurable



15

- L'architecture de machines HDFS repose sur des grappes. Un cluster inclut un noeud de nom (noeud maître) "NameNode" et plusieurs noeuds de données "dataNodes".

Exemple d'un grappe Hadoop cartographié sur le matériel



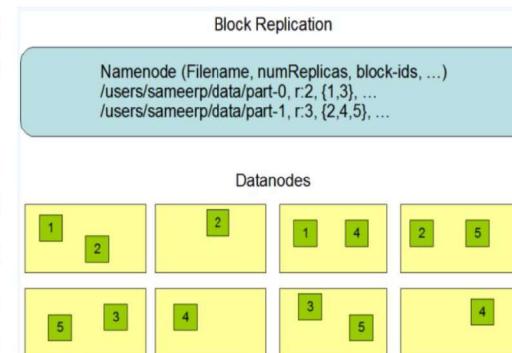
14

Nœud de nom (maître - Name Node)

- Le **noeud de nom** gère tous les **accès aux fichiers** y compris la **lecture, l'écriture, la création, l'effacement** et la **replication des blocs** de données sur **les noeuds de données**.
- Il maintient toutes les informations des blocs de données à l'aide de méta données

16

- Le **noeud de nom** doit savoir quels blocs sur quels noeuds de données construisent le fichier complet. Le **noeud de nom** utilise un “rack ID” pour garder trace des dataNodes dans le cluster.



17

Noeud de données (data node-workers)

- Les **noeud de données** fournissent continuellement des messages au **noeud de nom**
 - “heartbeat” (battement de Coeur) pour s’assurer du fonctionnement correct du **noeud de données**
 - ✓ Quand un **heartbeat** n'est plus présent, le **noeud de nom** fait sortir le **noeud de données du cluster** et continue à fonctionner comme si rien ne s'est passé.
- Ils fournissent un rapport de blocs “**BlockReport**” contenant la liste de blocs présents

19

- Le **noeud de nom** est un **point unique de défaillance**. Si ce service est arrêté, il n'yaura pas un moyen d'extraire les blocs d'un fichier.

→ Hadoop 2 a ajouté un **noeud de nom en veille (standby)** qui recevra un fichier de métadonnées **du noeud de nom actif (principal)** via le noeud de journal afin qu'en cas de défaillance du **noeud de nom** principal, le **noeud de nom** secondaire prend le relais.

18

Exemple d’écriture d’un fichier

Soit le fichier **monfichier.txt** de taille 160MB à stocker sous HDFS sous forme de blocs de 64 MB.

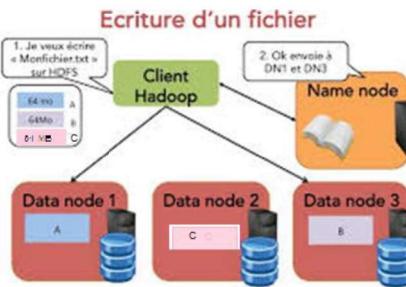
↓ **Solution?**

- Le programme du client Hadoop (un outil `hadoop` console) divise le fichier en blocs de 64 MB.
- $160/64= 2$ et reste 32MB.
=> 2 deux blocs de 64MB et un bloc de 32MB. Ce dernier occupera un espace de disque de 64MB.
=> **monfichier.txt** en blocs: 3 blocs: A, B et C

20

Exemple d'écriture d'un fichier

- Le **nœud de nom** sera interrogé pour sauvegarder les blocs du fichier
- Le **nœud de nom** informe le programme client des **nœuds de données** où seront enregistrés les blocs: bloc A dans **Data node 1** et blocs C et B dans **Data node 2 et 3** respectivement.
- Le fichier sera stocké par les **data nodes** et répliqué tout en informant le **nœud de nom**



21

- Les données qui sont **divisés en blocs, et réparties** sur plusieurs **nœuds de données** seront **traitées** avec deux tâches **map et reduce** développées par le programmeur.

→ Le même programme sera exécuté sur les nœuds de nom => plusieurs processus seront exécutés en même temps

Une fois que toutes les instances (nœuds) ont terminé les calculs, les résultats seront enregistrés localement.

23

MapReduce permet la **programmation distribuée** (Google 2004) des applications gourmandes en données d'une façon simplifiée pour le programmeur.

Ainsi, le modèle de programmation permet au développeur de ne s'intéresser qu'à la partie algorithmique. Il transmet alors son programme **MapReduce** développé dans un langage de programmation au **plateforme Hadoop** pour l'exécution.

22

- Fusionner les résultats partiels produits par des instances individuelles.

➤ « **job** » MapReduce concerne le travail que le client souhaite réaliser. Ce travail est constitué de données d'entrée (contenues dans HDFS), d'un programme MapReduce (implémentation des fonctions map et reduce) et de paramètres d'exécution.

24

On distingue 4 étapes dans un traitement mapReduce:

- map** {
 - **Diviser** les données d'entrées (split) sous forme de plusieurs fragments
 - **Mapper** chacun de ces fragments et obtenir, suite au traitement map, des couples <clé, valeur>
}

- Étape intermédiaire** {
 - **Grouper et trier** (shuffle and sort) ces couples <clé, valeur> par clé
}

- reduce** {
 - **Réduire**: récolter les résultats partiels des mappers pour avoir le résultat finale avec le traitement de « reduce »
}

25

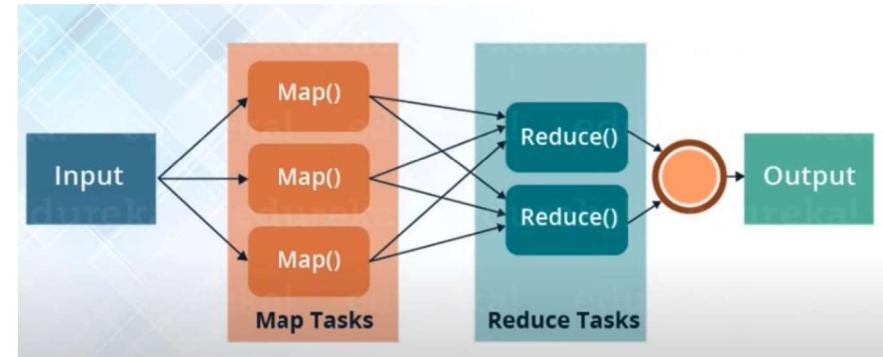
Exemple: Un utilisateur définit une fonction pour une application qui compte le nombre d'occurrences de chaque mot dans un ensemble de documents

Supposons que le traitement se fait les documents suivants:

Apple Orange Mango
Orange Grapes Plum

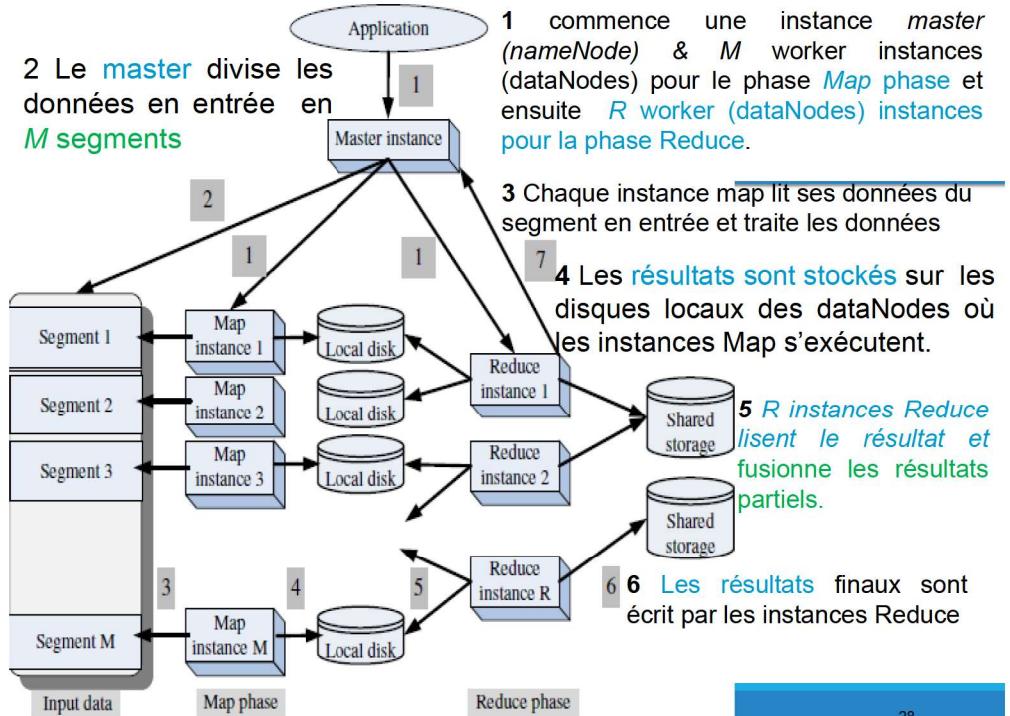
Apple Plum Mango
Apple Apple Plum

27



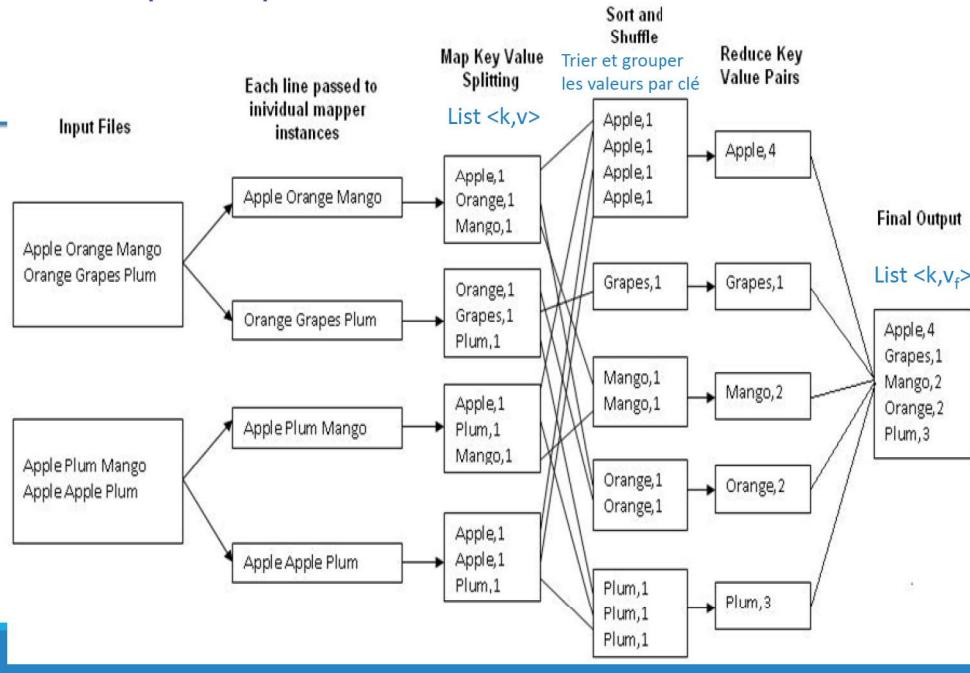
Edureka, Hadoop certification training

26



28

Processus pour compter les nombre de mots



Ecriture d'un programme mapReduce pour l'exemple: compter les mots

La programmation mapReduce peut être effectuée avec plusieurs langages tels que JAVA, python, R, etc. Dans ce cours, la programmation sera en python.

Ecriture du programme mapReduce en Python et exécution en local

Mapper Class

- Chaque entrée sera générée en <clé, valeur> suivant le traitement map.

Composants principaux de Hadoop

Le modèle de programmation MapReduce

Exemple:

mapper

Apple Apple plum
Apple:1 Apple:1 plum: 1 ...

Grouper et trier

Apple:1,1 plum:1

reducer

Apple:2 plum:1

30

Ecriture d'un programme mapReduce pour l'exemple:
compter les mots

Code du mapper

```
def mapper(self, key, line):
    words = line.split(' ')
    for word in words:
        yield word.lower(), 1
```

```
def mapper(self, key, line):
    words = line.split(' ')
    for word in words:
        yield word.lower(), 1
```

- **L'entrée:**

- La **clé** (key) n'est rien, mais le décalage de chaque ligne dans le fichier texte



- **La sortie :**

- La **clé** qui représente les mots segmentés
- La **valeur** : assigner 1 pour chaque mot => la valeur est 1

Exemple :
Apple 1,
Plum 1,
etc.

- Les instructions dans la méthode map permettent de faire la segmentation, de chaque ligne en entrée, en mots à l'aide de **Split**. Assigner à chaque mot (word) la valeur 1.

33

Ecriture d'un programme mapReduce pour l'exemple:
compter les mots

Reducer Class

- Les sorties générées par le mapper ,passent par une étape intermédiaire automatique **shuffle and sort**: grouper les valeurs suivant les clés et les trier
- les valeurs résultats de **shuffle and sort** sont fournis au reducer, qui va les traiter et générer les résultats finaux.

34

Ecriture d'un programme mapReduce pour l'exemple:
compter les mots

Code du reducer

```
def reducer(self, key, values):
    yield key, sum(values)
```

- **L'entrée:**

- La **clé** la liste de mots obtenus après l'opération **sort and shuffle** (trier et grouper: opération effectuée automatiquement après l'exécution du mapper)

Exemple :
Apple 1, 1,1,1
Plum 1,1,1
etc.

- **La sortie :**

- La **clé** est les mots présents dans le fichier en entrée
- La **valeur** : la somme de la liste de 1 pour chaque mot => nombre d'occurrences de chaque mot.

35

Ecriture d'un programme mapReduce pour l'exemple:
compter les mots

Programme complet

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        words = line.split(" ")
        for word in words:
            yield word.lower(), 1
    # On ne peut pas écrire: yield words.lower(), 1
    # sans la boucle car words est une liste de mots
    def reducer(self, key, values):
        yield key, sum(values)

    if __name__ == '__main__':
        MRWordFrequencyCount.run()
```

- La classe mrjob permet de faciliter la programmation avec python et exécuter map et reduce à partir d'un seul fichier.

- Un Job est défini comme une classe qui hérite de la classe MRJob. Cette classe contient les méthodes qui définissent les étapes du job

36

Ecriture d'un programme mapReduce pour l'exemple: compter les mots

Programme complet

```
from mrjob.job import MRJob  
  
class MRWordFrequencyCount(MRJob):  
  
    def mapper(self, _, line):  
        words = line.split(" ")  
        for word in words:  
            yield word.lower(), 1  
  
    # On ne peut pas écrire: yield words.lower(), 1  
    # sans la boucle car words est une liste de  
    # mots  
  
    def reducer(self, key, values):  
        yield key, sum(values)  
  
if __name__ == '__main__':  
    MRWordFrequencyCount.run()
```

- Un step se constitue d'un mapper, un combiner et un reducer. Aucun n'est obligatoire

- Ces deux dernières lignes sont obligatoires pour dire que MRWordFrequencyCount et le job

37

Ecriture du programme mapReduce en Python et exécution en distribué

L'exécution sur windows en local était permise avec Mrjob qui as permis le mapper et le reducer dans un seul fichier.

Pour l'exécution sous Linux ou sur Hadoop sans utiliser MRJob, il faut avoir un fichier pour le mapper et un fichier pour le reducer et il faut faire plus de précision dans le code, qui sera plus compliqué sans MRJob car il faut fournir tous les détails de lecture, d'écriture, des exceptions etc.

```
#!/usr/bin/env python  
"""mapper.py"""  
  
import sys  
  
# lire les entrées à partir de l'entrée standard STDIN (standard input)  
for line in sys.stdin:  
    # enlever les espaces au début et à la fin de chaque ligne (line)  
    line = line.strip()  
  
    # split: segmenter line en mots => words est une liste qui contient les  
    # mots  
  
    words = line.split()  
  
    # parcourir la liste et incrémenter le compteur  
    for word in words:  
        # écrire les résultats à la sortie standard STDOUT (standard output);  
        # les résultats sont délimités avec le signe tabulation, pour chaque mot  
        # attribuer 1  
  
        print '%s\t%s' % (word.lower(), 1)
```

39

La sortie du mapper est toujours l'entrée du reducer après l'opération **shuffle and sort**.

```
#!/usr/bin/env python  
"""reducer.py"""  
from operator import itemgetter  
import sys  
current_word = None  
current_count = 0  
word = None  
  
# les entrées arrivent de STDIN  
for line in sys.stdin:  
    # supprimer les espaces de début et de fin de ligne  
    line = line.strip()  
    # segmenter les entrées issu du mapper.py  
    word, count = line.split("\t", 1)  
    # convertir count (qui est String) en int  
    try:  
        count = int(count)  
    except ValueError:  
        # si count n'est pas un nombre, ignore et passer la ligne  
        continue
```

40

```
# Ce if fonctionne car Hadoop tri les résultats de map par clé (ici: mot) avant de les
# passer au reducer
if current_word == word:
    current_count += count
else:
    if current_word:
        # écrire les résultat à STDOUT
        print '%s\t%s' % (current_word, current_count)
current_count=count
current_word=word

# ne pas oublier le résultat du dernier mot si nécessaire
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
#%s => convertir la valeur en chaîne
```