Bonjour

Introduction au big data

Stockage distribué

Ce premier cours présente les concepts suivants

- ➔ Introduction
  - ◆ C'est quoi le big data ?
  - ◆ Comment y faire face ?
- ➔ Hadoop
- ➔ Systèmes de fichiers distribués (HDFS)
- ➔ Hand-on lab Hadoop
  - ◆ Mettre en place un cluster hadoop en mode local
  - ◆ Se familiariser avec les commandes de bases
  - ◆ Programmation API pour HDFS

EPITA

Pourquoi ce cours ?

Un domaine très prisé par les entreprises

Selon LinkedIn, les compétences les plus recherchées depuis plusieurs années sont :

Data Science

Cloud and Distributed Computing

Statistical Analysis and Data Mining

Storage Systems and Management

Avant de parler de BigData, connaissez vous les préfixes ?

Unité	Abbréviation	Taille en octets	Exemple
1 octet		1	4 octets ≃ Un caractère au format utf-8
1 kilooctet	ko	10^3	20 ko ≃ Mots prononcés par une personne, par jour
1 mégaoctet	Mo	10^6	3 Mo ≃ Texte d'une édition quotidienne du New York Times
1 gigaoctet	Go	10^9	1 Go ≃ Mots prononcés par une personne durant une vie
1 téraoctet	To	10^{12}	593 To ≃ Mots prononcés par l'ensemble de la population mondiale, par jour
1 pétaoctet	Po	10^{15}	4 Po ≃ Données générées par Facebook, par jour
1 exaoctet	Eo	10^{18}	605 Eo ≃ Données générées par le téléscope SKA, par jour
1 zétaoctet	Zo	10^{21}	40 Zo ≃ Données générées par le web en 2020, par an

EPITA

Comment définiriez-vous le Big Data ?

- A partir de quel moment parle t-on de big data ?
- Faut t'il plus de disques ? plus de machines ?
- Comment analyser une grosse quantité de données ?

Big Data ?

➔ Des collections de données gigantesques, impossible à stocker et à traiter sur nos machines locales

- Internet : Google en 2015 : 10 Eo (10 milliards de Go), Facebook en 2018 : 1 Eo de données, 7 Po de nouvelles données par jour, Amazon : 1 Eo.

➔ Tout et n'importe quoi est stocké, dans l'idée que ça pourrait être exploité un jour.

Les 5 V’s du Big Data

Volume

Volume = size :

Chaque minute

- ➔ 204 million d'emails envoyés
- ➔ 200,000 photos et 1.8 million de likes sur facebook
- ➔ 1.3 millions de vues sur Youtube

Variety = complexity :

différents types de données

- ➔ Image
- ➔ Texte
- ➔ Graphe
- ➔ Signal

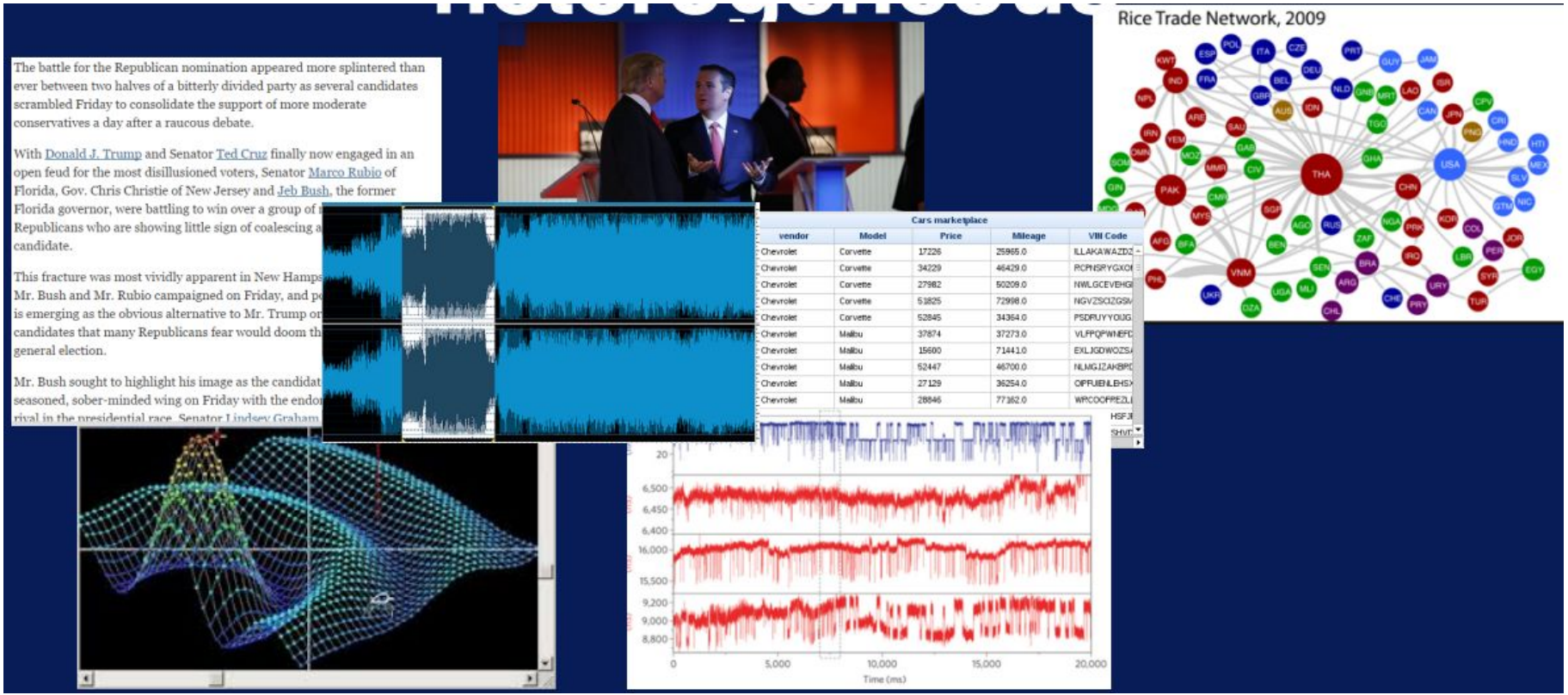
Variety

Il était une fois ... les bases de données relationnelles

Cars marketplace				
vendor	Model	Price	Mileage	VIN Code
Chevrolet	Corvette	17226	25965.0	ILLAKAWAZDZ
Chevrolet	Corvette	34229	46429.0	RCPNSRYGXOI
Chevrolet	Corvette	27982	50209.0	NWLGCEVEHGI
Chevrolet	Corvette	51825	72998.0	NGVZSCIZGSM
Chevrolet	Corvette	52845	34364.0	PSDRUYYOIJG.
Chevrolet	Malibu	37874	37273.0	VLFPQPWNEFD
Chevrolet	Malibu	15600	71441.0	EXLJGDWOZSA
Chevrolet	Malibu	52447	46700.0	NLMGJZAKBRD
Chevrolet	Malibu	27129	36254.0	OIPFUIENLEHSX
Chevrolet	Malibu	28846	77162.0	WRCOOFREZLL
Chevrolet	Malibu	46165	60590.0	HUFTTHQHSFJF
Chevrolet	Malibu	18263	37790.0	JLMHNAFSHVD

EPITA

Aujourd'hui, ça ressemble plus à ça



Pensez à une grosse collection de mails

C’est un mélange de :

➔ Données structurées (expéditeur, date, etc)
➔ Données non structurées (Corps du mail, l’objet, etc)
➔ Données multimédias (pièces jointes, etc)

Velocity

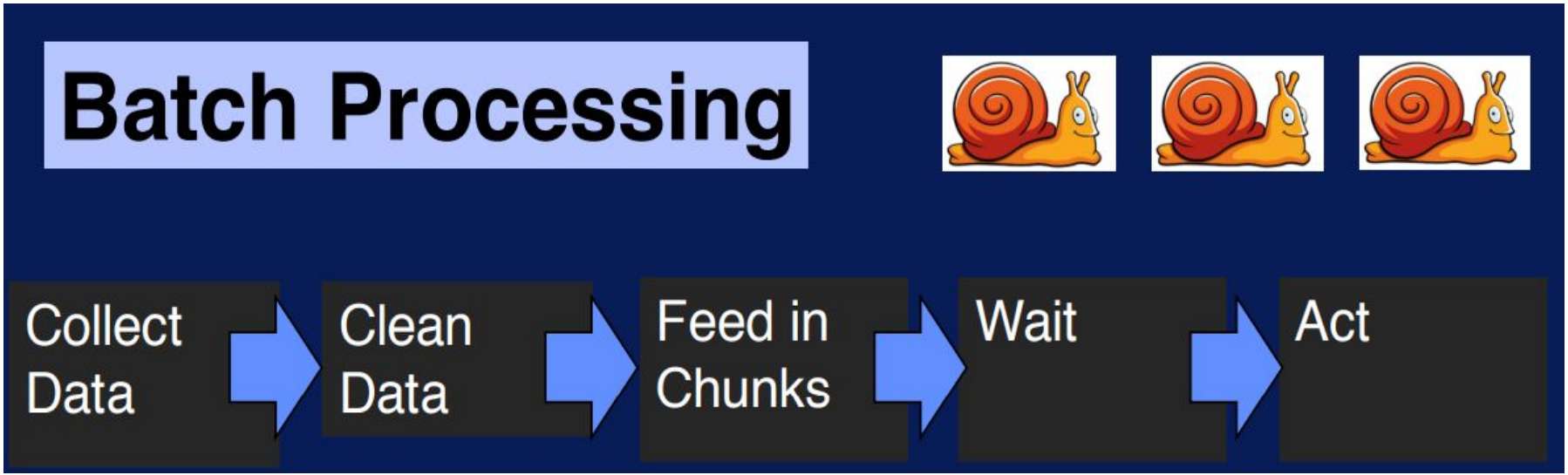
Velocity = speed:

rapidité d’évolution des données

Besoin de traitement temps réel

➔ Plateforme de streaming
➔ Transactions financières

Traitement par lots (Batch)



Traitement temps réel

Real-Time Processing

Instantly capture streaming data → Feed real time to machines → Process Real Time → Act

EPITA

Veracity = qualité :

- ➔ La pertinence des données (erreur, duplication, inconsistence)
- ➔ Incertitude
- ➔ Faire attention aux erreurs

Veracity

Value : Getting out value from Big Data

- ➔ Big Data-> Analyze->Insight-> Action
- ➔ Historique des données -> Prédictions

Exemples

- ➔ Recommandations
- ➔ Market Basket Analysis
- ➔ Détection de pannes

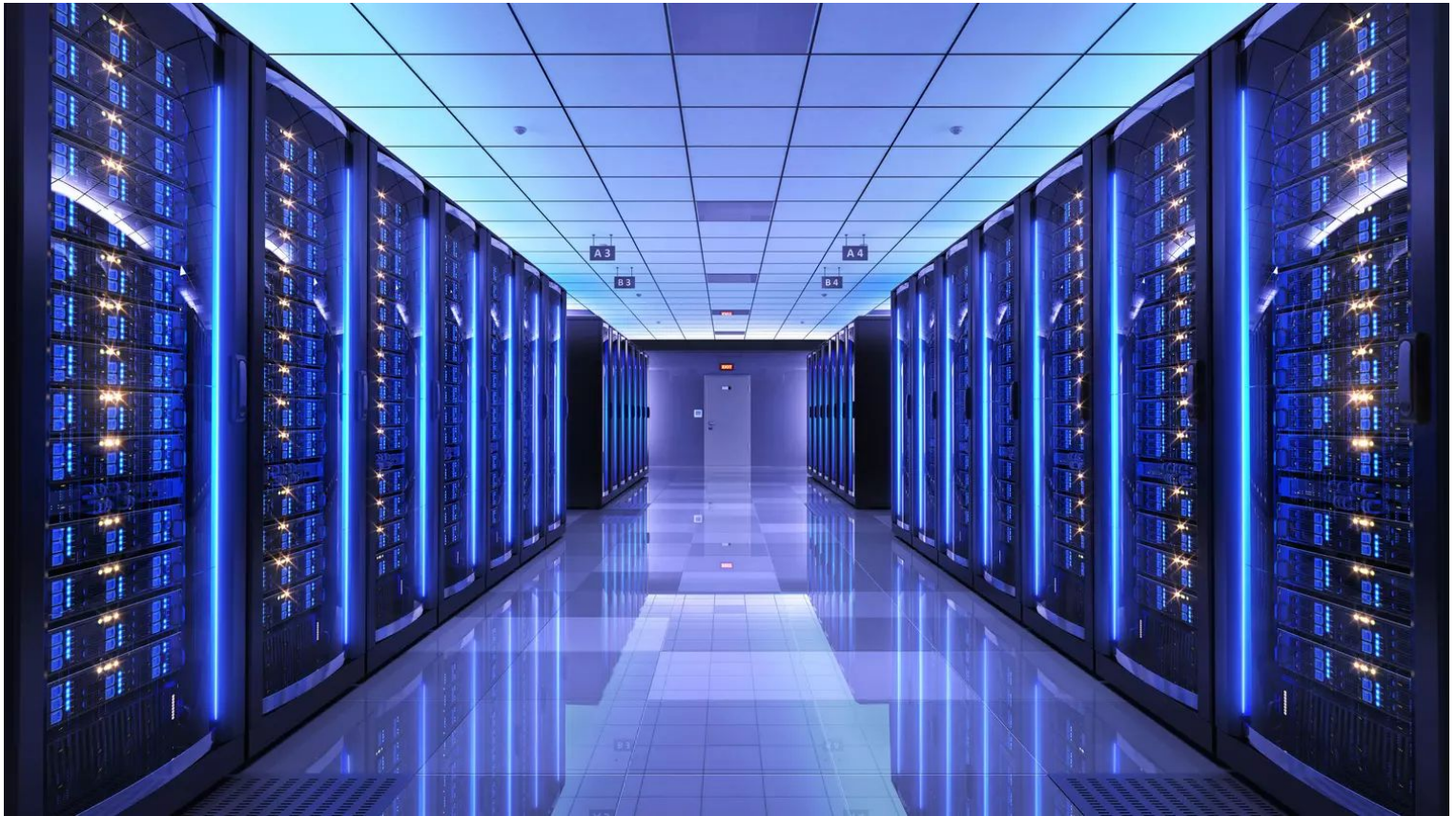
Value

Comment faire face à cette problématique de données gigantesques?

Distribution des données et leurs traitements

- Le traitement d'aussi grandes quantités de données impose des méthodes particulières.
- Un SGBD classique, même haut de gamme, est dans l'incapacité de traiter autant de données
- Répartir les données sur plusieurs machines (jusqu'à plusieurs millions d'ordinateurs) dans des Data Centers
- système de fichiers spécial permettant de ne voir qu'un seul espace pouvant contenir des fichiers gigantesques
- Traitements du type « map-reduce » : algorithmes faciles à écrire, exécutions faciles à paralléliser.

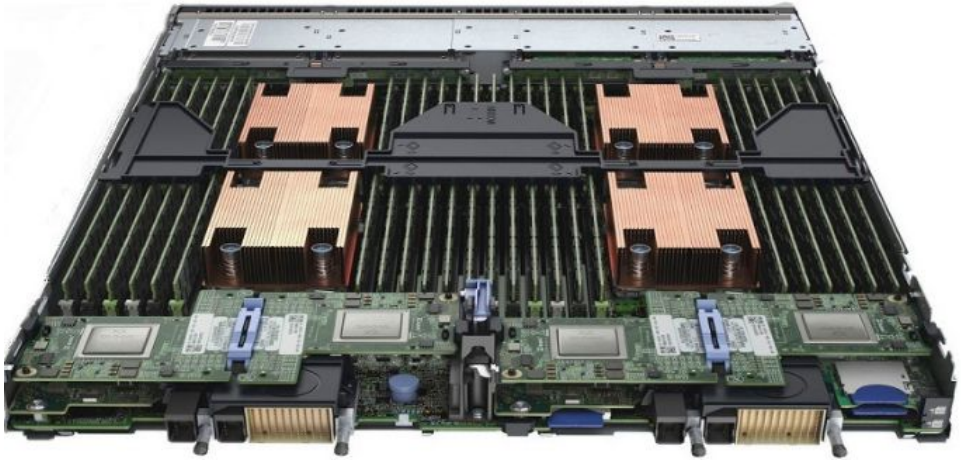
Imaginez 5000 ordinateurs connectés entre eux formant un data center



EPITA

Un très grand nombre de machines connectées

Chacun de ces racks (blade computer) ou rack server peut ressembler à ceci (par exemple : 4 CPU multi-cœurs, 1 To de RAM, 24 To de disques rapides, 5000e, prix et technologie en constante évolution) :



- Toutes ces machines sont connectées entre elles afin de partager l'espace de stockage et la puissance de calcul.
- Le Cloud est un exemple d'espace de stockage distribué
- L'exécution des programmes est également distribuée

Stockage distribué

Hadoop

La petite histoire de Hadoop

L'anecdote dit que Hadoop, au départ, c'est le nom de ce petit éléphant en peluche. Appartenant au fils de l’un des créateurs du framework Hadoop

EPITA
ÉCOLE D'INGENIEURS EN INFORMATIQUE

Hadoop (historique)

En 2002, Doug Cutting et Mike Cafarella, deux ingénieurs, décident de s'attaquer au passage à l'échelle de Lucene, le moteur de recherche open source

EPITA

Socle Technique de Hadoop

Le socle technique d'Hadoop est composé :

- De toute l'architecture support nécessaire pour l'orchestration, c'est-à-dire :
  - l'ordonnancement des traitements,
  - la localisation des fichiers,
  - la distribution de l'exécution.
- D'un système de fichiers **HDFS** qui est :
  - Distribué : les données sont réparties sur les machines du cluster.
  - Répliqué : en cas de panne, aucune donnée n'est perdue.
  - Optimisé pour la colocalisation des données et des traitements.

Hadoop Distributed File System (HDFS)

Présentation de HDFS

- ➔ HDFS est un système de fichiers distribué.
  - ◆ Les fichiers et dossiers sont organisés en arbre (comme Unix)
  - ◆ Ils sont stockés sur un grand nombre de machines
  - ◆ La position exacte d'un fichier est invisible, l'accès aux données est transparent
- ➔ Les fichiers sont copiées en plusieurs exemplaires
- ➔ HDFS permet de voir tous les dossiers et fichiers de ces milliers de machines comme un seul arbre

Organisation des fichiers

➔ Vu de l'utilisateur HDFS ressemble à un système de fichiers Unix

- Il y a une racine, des répertoires et des fichiers
- Les fichiers ont un propriétaire, un groupe et des droits d'accès

Commandes hdfs dfs

- ➔ hdfs dfs -help
- ➔ hdfs dfs -ls
- ➔ hdfs dfs -cat fichier
- ➔ hdfs dfs -mv ancien nouveau
- ➔ hdfs dfs -cp ancien nouveau
- ➔ hdfs dfs -mkdir dossier
- ➔ hdfs dfs -rm -f -r

Échanges entre HDFS et votre machine locale

Pour placer un fichier dans HDFS depuis votre machine locale, deux commandes équivalentes

- ➔ hdfs dfs -copyFromLocal fichier_source fichier_destination
- ➔ hdfs dfs -put fichier_source fichier_destination

Pour extraire un fichier HDFS vers votre machine locale, deux commandes équivalentes

- ➔ hdfs dfs -copyToLocal fichier_source destination
- ➔ hdfs dfs -get fichier_source destination

Comment fonctionne HDFS

- ➔ Chaque fichier HDFS est découpé en blocs de taille fixe
- ➔ Souvent un bloc HDFS=64Mo ou 128Mo (c'est configurable)
- ➔ Selon la taille du fichier, il lui faudra un certain nombre de blocs (le dernier bloc d'un fichier fait la taille restante)
- ➔ Les blocs d'un fichier sont stockés sur différents machines (les data nodes)
- ➔ Chaque bloc est répliqué sur plusieurs machines
  - ◆ Cela permet d'y accéder simultanément par plusieurs processus
  - ◆ Permet aussi de se prémunir contre les pannes

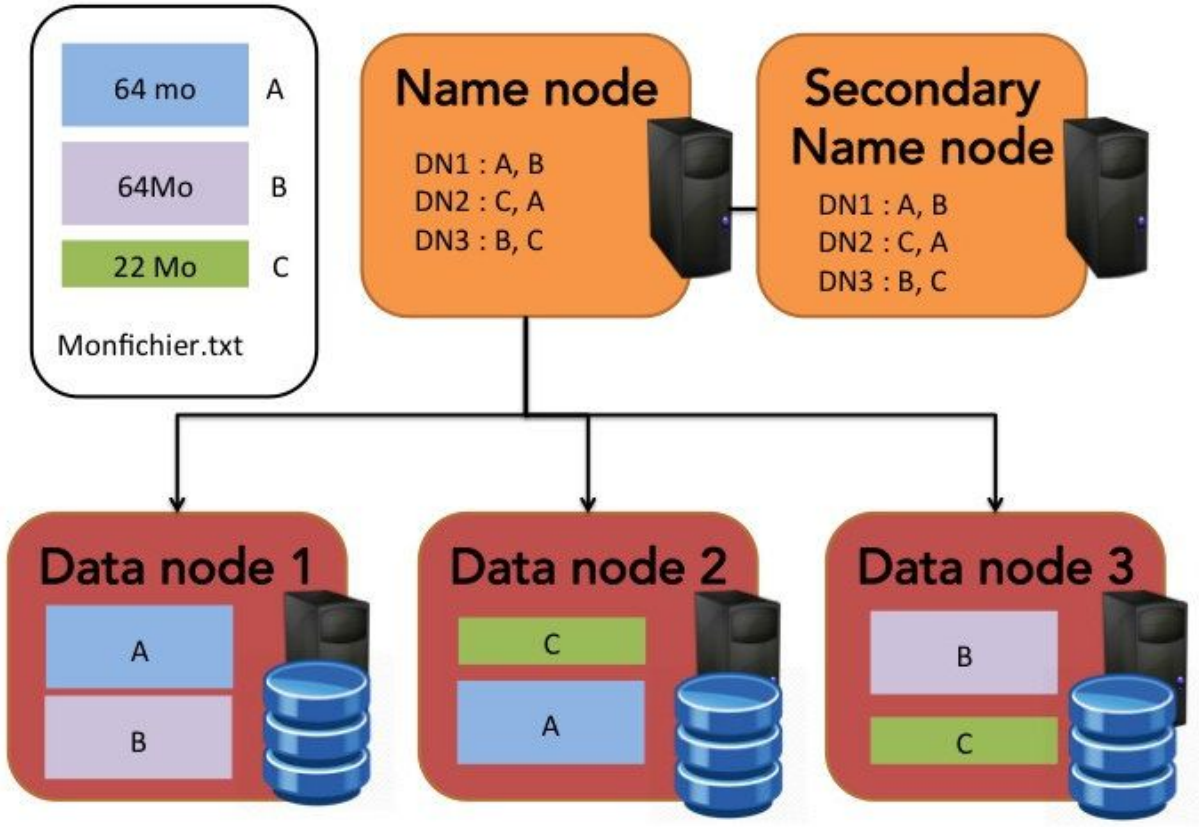
Organisation des machines pour HDFS

Un cluster HDFS est constitué de machines jouant différents rôles. Il s'agit d'une architecture maître-esclave

- Le nœud maitre appelé **name node** contient et stocke tous les noms et blocs des fichiers ainsi que leur localisation dans le cluster. On peut donc le voir comme un gros annuaire
- Une autre machine, appelée **secondary name node** sert de namenode de secours en cas de défaillance du nœud maître et il a donc pour rôle de faire des sauvegardes régulières de l'annuaire.
- Les autres nœuds, les esclaves, sont les nœuds de stockage en tant que tels. Ce sont les **data nodes** qui ont pour rôle la gestion des opérations de stockage locales (création, suppression et réplication de blocs) sur instruction du name node.

EPITA

Organisation des machines pour HDFS



Explications

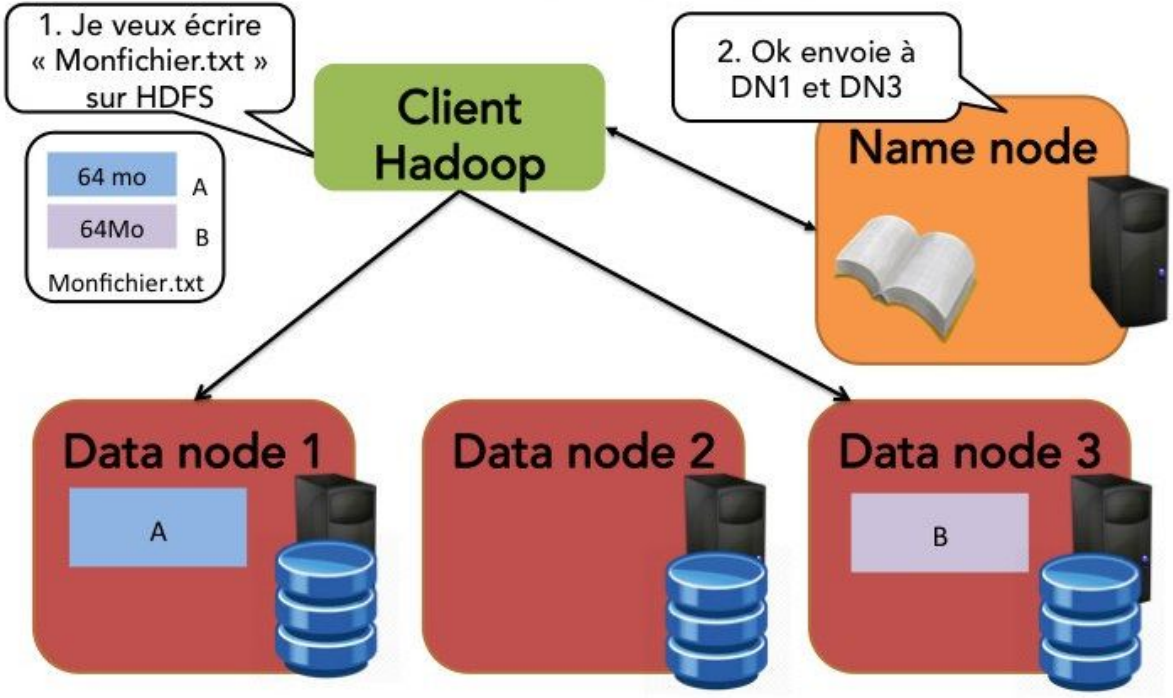
Les datanodes contiennent des blocs. Les mêmes blocs sont dupliqués (réplication) sur différents datanodes, en général 3 fois (c'est configurable). Cela assure :

- Fiabilité des données si une machine tombe malade
- Accès parallèle par différents processus aux mêmes données

Le namenode (le maître) connaît :

- Sur quels blocs sont contenus les fichiers
- Sur quels datanodes se trouvent les blocs en question
- Je vous rassure : ce n'est pas à nous utilisateurs qui vont s'occuper ni de découper les données en blocs ni de savoir ou ils se trouvent

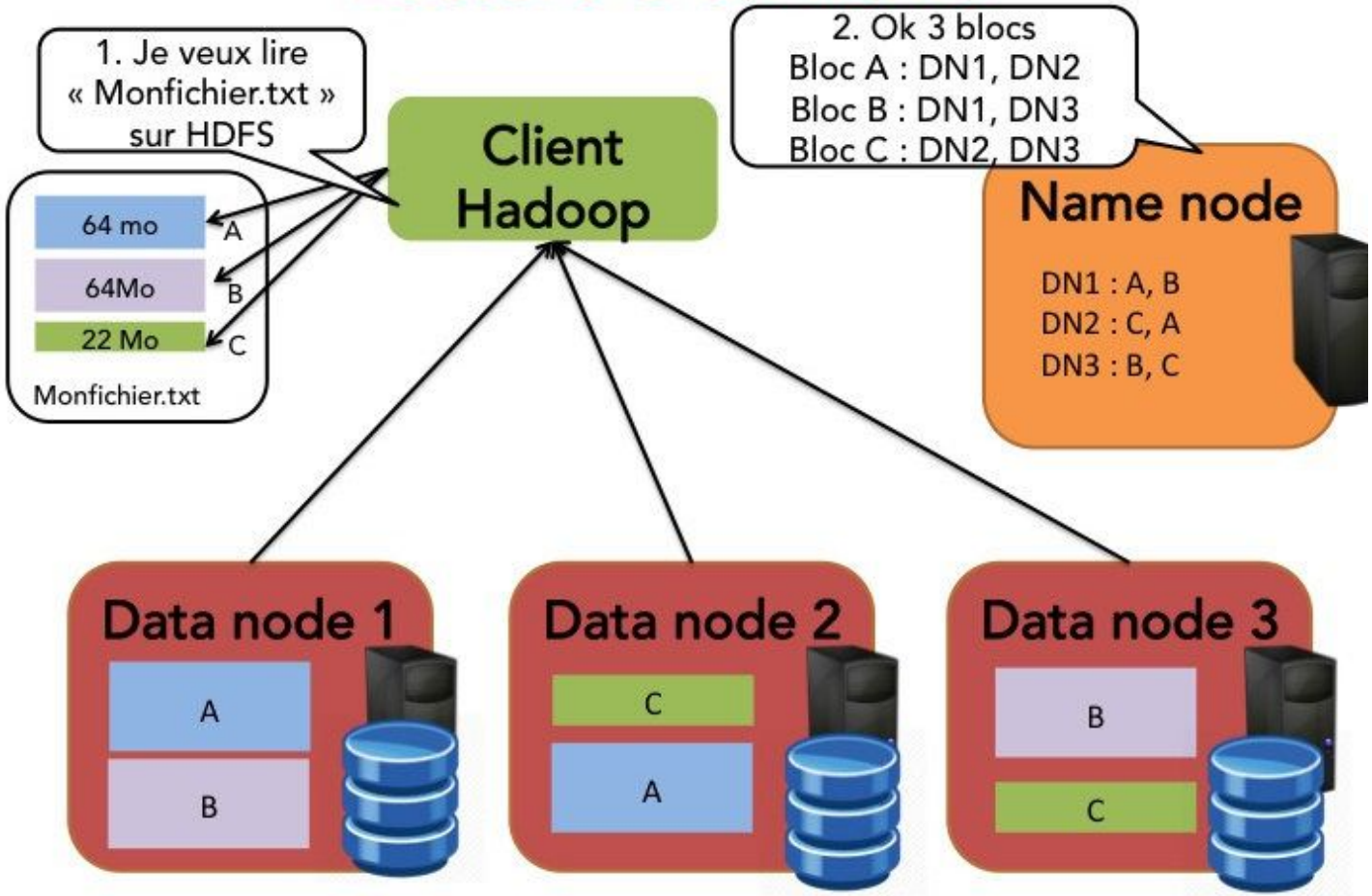
Ecriture d'un fichier



Ecriture d’un fichier

1. Le client indique au name node qu'il souhaite écrire un bloc.
2. Le name node indique le data node à contacter.
3. Le client envoie le bloc au data node.
4. Les data nodes répliquent les blocs entre eux.
5. Le cycle se répète pour le bloc suivant.

Lecture d'un fichier



Lecture d’un fichier

1. Le client indique au name node qu'il souhaite lire un fichier.
2. Le name node indique sa taille ainsi que les différents data nodes contenant les blocs.
3. Le client récupère chacun des blocs sur l’un des data nodes.
4. Si le data node est indisponible, le client en contacte un autre.

API Python Pour HDFS (TP)

Modèles de programmation large échelle (Map-Reduce)

Inspiré largement du paradigme diviser pour régner

- Diviser : découper le problème initial en sous problèmes
- Régner : résoudre les sous-problèmes indépendamment
- Combiner : construire la solution du problème initial en combinant les solutions des différents sous problèmes

➔ Mapreduce c'est **Diviser pour distribuer pour régner**

➔ Les données sont distribuées sur plusieurs machines,

Inspiration fonctionnelle

- **map** : consiste à appliquer une même fonction à tous les éléments de la liste

  map(f)[x0, ..., xn] = [f(x0, ...,f(xn)]

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

- **reduce** applique une fonction récursivement à une liste et retourne un seul résultat

  reduce(f)[x0, ..., xn] = f(x0, f(x1, f(x2, ...)))

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

Les données sont toujours représentées par des paires (clé, valeur)

- L'ensemble des données à traiter est découpé en plusieurs lots ou sous-ensembles.
- Dans une première étape, l'étape MAP, est appliquée à chaque lot. Cette opération transforme la paire (clé, valeur) représentant le lot en une liste de nouvelles paires (clé, valeur)constituant ainsi des résultats intermédiaires du traitement à effectuer sur les données complètes.
- Avant d'être envoyés à l'étape REDUCE, les résultats intermédiaires sont regroupés et triés par clé. C'est l'étape de SHUFFLE and SORT.
- Enfin, l'étape REDUCE consiste à appliquer l'opération reduce, spécifiée pour notre problème, à chaque clé. Elle agrège tous les résultats intermédiaires associés à une même clé et renvoie donc pour chaque clé une valeur unique.

EPITA

WordCount le helloworld de Map-Reduce

Word Count !

Compter le nombre d'occurrences de chaque mot dans un ensemble de textes.

Données : un ensemble de textes

Le jour se lève sur notre grisaille, sur les trottoirs de nos ruelles et sur nos tours
[...]

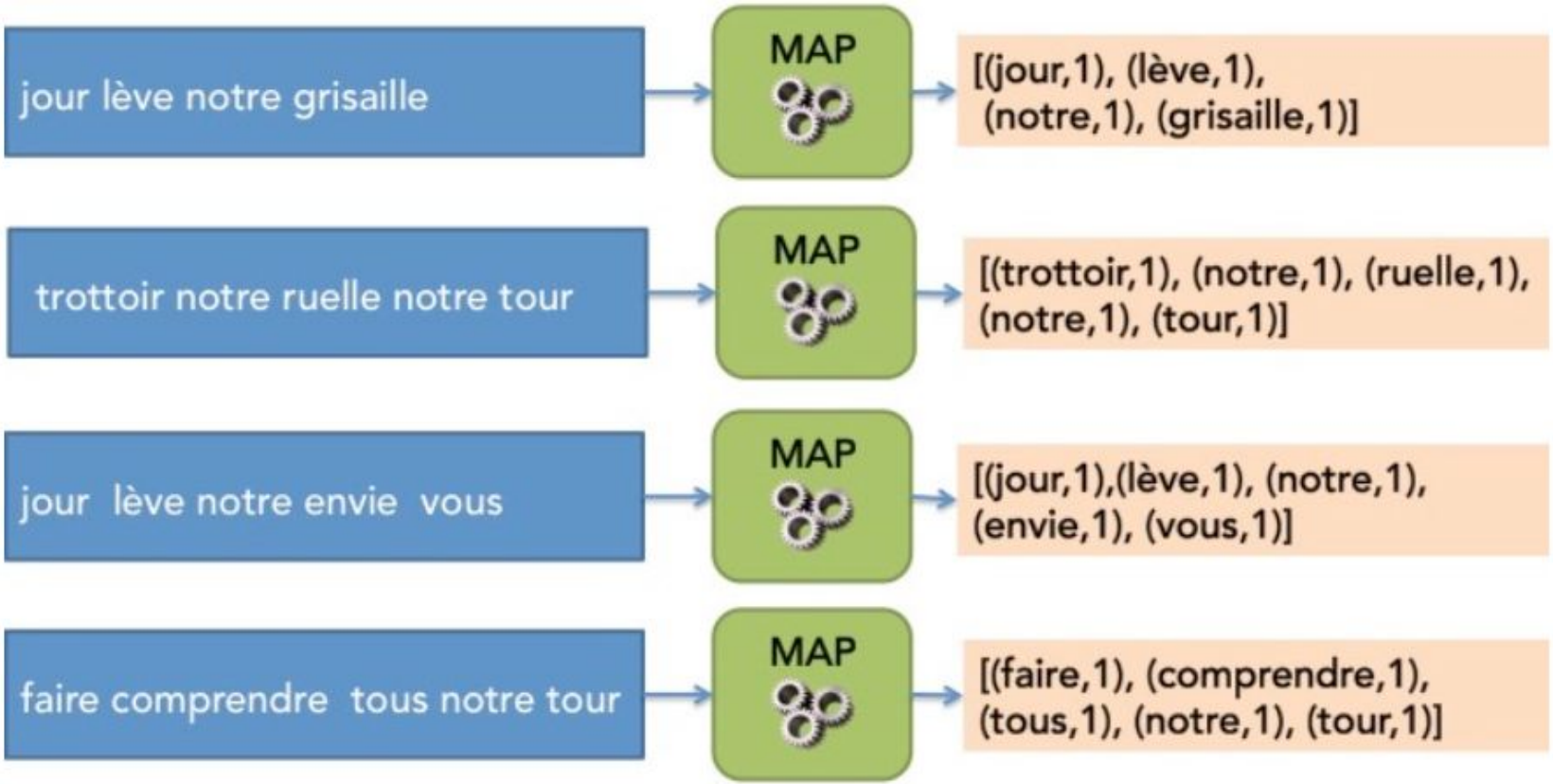
Le jour se lève sur notre envie de vous faire comprendre à tous que c'est à notre tour
[...]

EPITA

Le Mapper

```
def map(key,value):
    intermediate=[]
    for word in value.split():
        intermediate.append((word, 1))
    return intermediate
```

Résultats


jour lève notre grisaille
MAP
[(jour,1), (lève,1), (notre,1), (grisaille,1)]
trottoir notre ruelle notre tour
MAP
[(trottoir,1), (notre,1), (ruelle,1), (notre,1), (tour,1)]
jour lève notre envie vous
MAP
[(jour,1),(lève,1), (notre,1), (envie,1), (vous,1)]
faire comprendre tous notre tour
MAP
[(faire,1), (comprendre,1), (tous,1), (notre,1), (tour,1)]

Shuffle and Sort

(comprendre, [1])

(envie,[1])

(faire,[1])

(grisaille,[1])

(jour,[1,1])

(lève, [1,1])

(notre, [1,1,1,1,1])

(ruelle,[1])

(tour,[1,1])

(tous,[1])

(trottoir,[1])

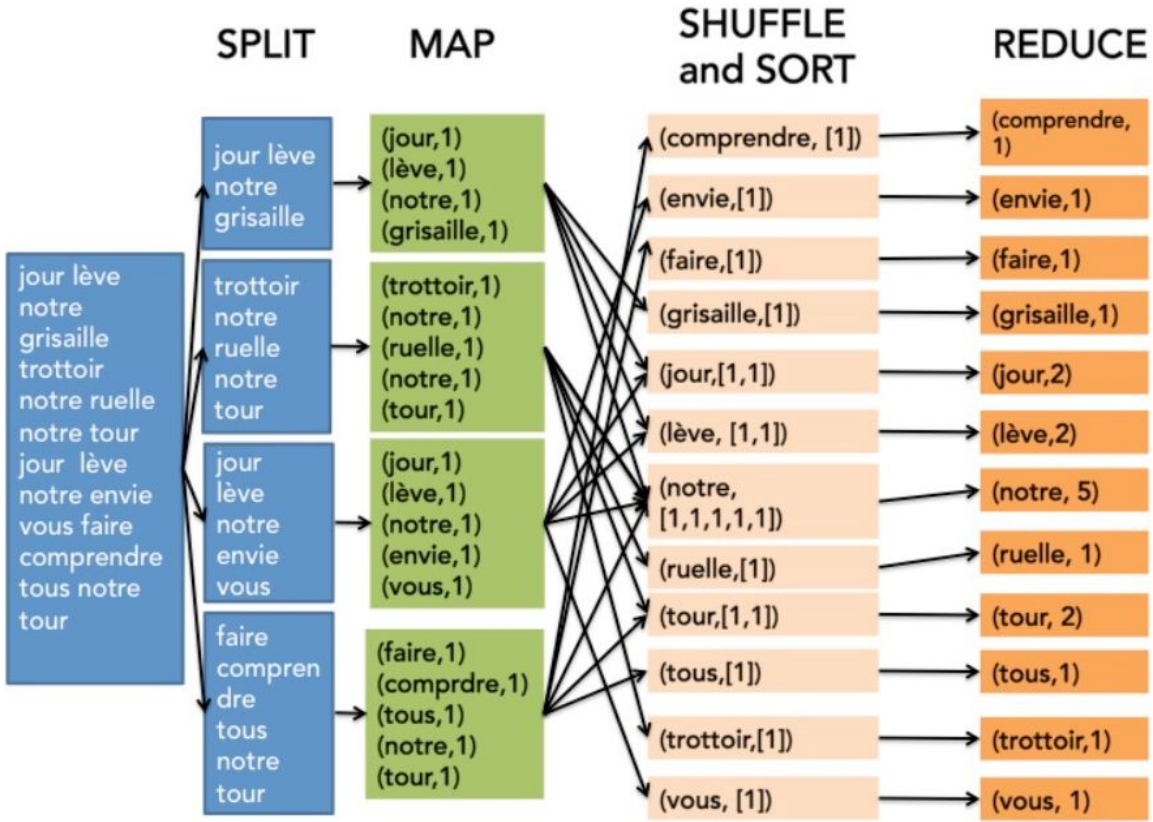
(vous, [1])

EPITA

Le reducer

```
def reduce(key, values):
    result = 0
    for c in values:
        result = result +c
    return (key, result)
```

The Big Picture



EPITA

YARN

Qu’est ce que YARN

YARN (Yet Another Resource Negociator) est un mécanisme dans Hadoop permettant de gérer des jobs sur un cluster

Il permet aux utilisateurs de lancer des jobs MapReduce sur des données présentes dans HDFS

Suivre leur avancement, récupérer des messages (logs)

Peut déplacer un processus d’une machine à une autre en cas de défaillance

YARN est transparent pour l’utilisateur. On lance l'exécution d’un programme MapReduce et YARN s’assure qu’il soit exécuté le plus rapidement possible

EPITA

Biblio et lectures

- https://openclassrooms.com/fr/courses/4297166-realisez-des-calculs-distribues-sur-des-donnees-massives

Merci !


EPITA
ÉCOLE D'INGENIEURS EN INFORMATIQUE