Fouille de données et *big data*

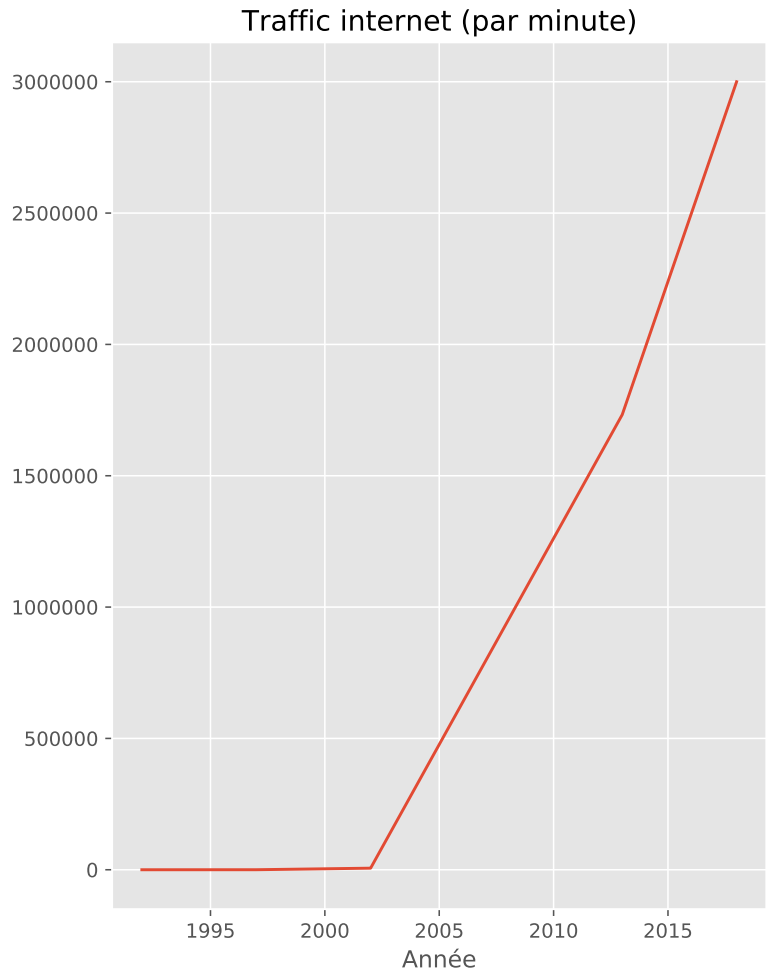
Maximilien Servajean

2019-2020

Plan

Les défis du *big data*

- Les 3**V** : Volume, Variété, Vélocité (Véracité, Valeur, Visibilité, etc.)
- 2500000000000000000 octets de données créés chaque jour
- 204000000 messages envoyés chaque minute
- 72 heures de vidéos sont uploadées sur Youtube chaque minute
- ...



`http://www.markentive.fr/wp-content/uploads/2016/03/infographie_big_data_web.jpg`

Les opportunités du *big data*

- Recommandation de produits
- Maison et villes connectées
- Améliorer les services clients
- Voitures connectées
- Agriculture connectée
- Amélioration des transports publics
- Gestion des déchets
- Santé
- Analyses de données publiques
- etc.

Fouille de données et *big data*

Quelles stratégies mettre en place lorsqu'on est face à une grande masse de données ?

- Réduction de la taille et/ou de la dimension du problème :
  - échantillonage,
  - indexation, KNN,
  - ACP, etc.
- Utilisation du calcul parallèle :
  - distribution, réplication, etc.

Spark

Spark est un *framework* de calcul distribué.Il permet de déployer l'exécution du code sur des *clusters*ăentiers sans avoir à expliciter la parallélisation.
Spark s'accompagne :

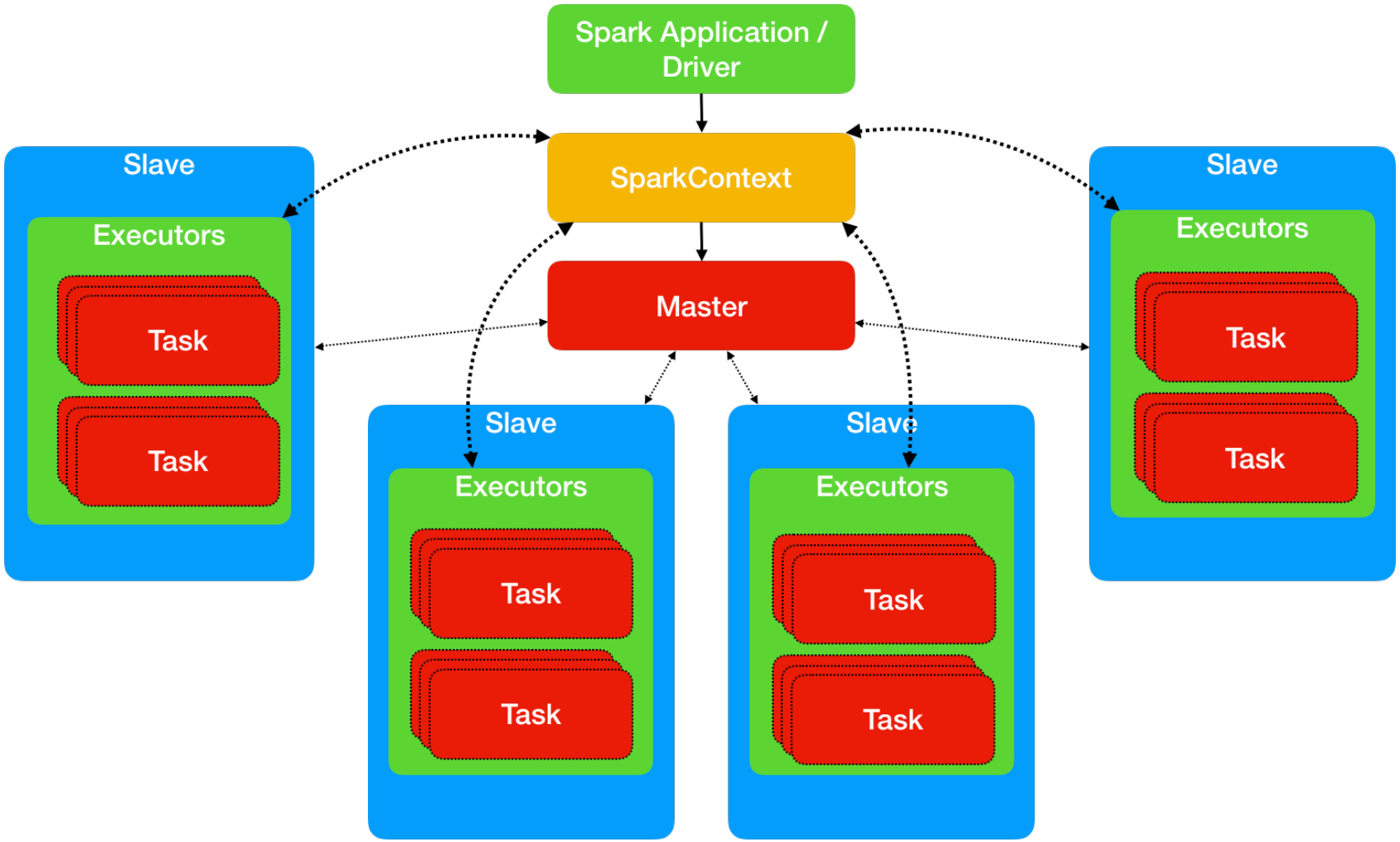
1. d'un gestionnaire de cluster (e.g. YARN),
2. d'un système de fichiers distribué (e.g. Hadoop Distributed File System).

Spark

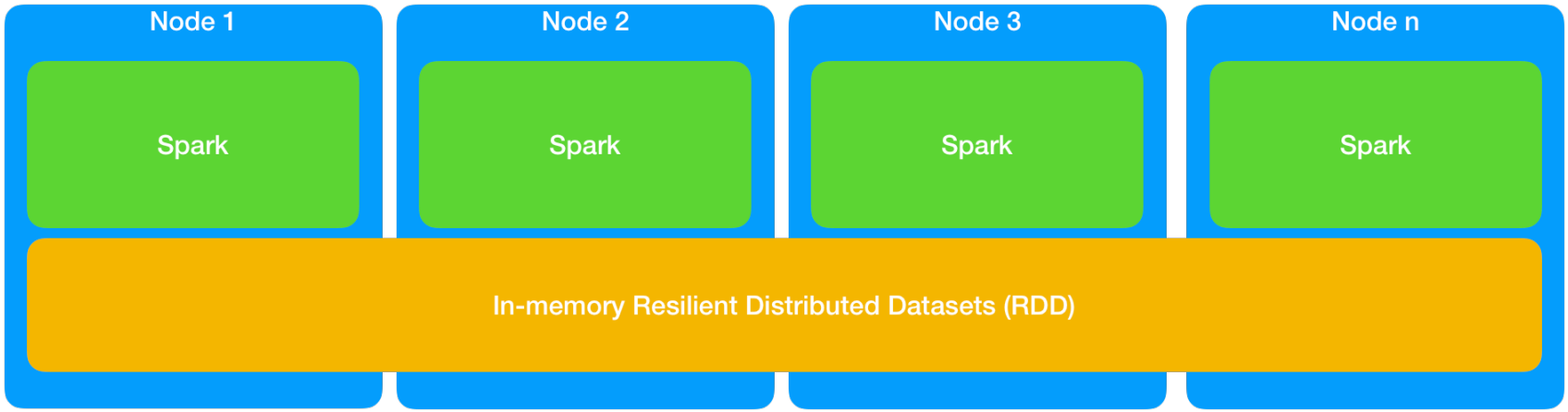
- Projet initié à l'université de Californie Berkeley
- API centrée autour de l'idée de RDD (`Resilient Distributed Dataset`), dataframes et datasets
- Un RDD décrit une série de transformations sur les données
- La distribution est invisible pour l'utilisateur
- Les RDDs sont tolérants aux fautes
- Spark est pensé en priorité pour Scala mais fonctionne aussi avec d'autres langages : R, Java, Python

Plan

Architecture de Spark (1/2)



Architecture de Spark (2/2)



- Le RDD est la plus ancienne structure utilisée par Spark
- Les RDDs sont immutables et sont créés lorsque l'on
  - charge des données,
  - transforme des données.
- Les RDDs ne sont pas des données, il s'agit du pipeline décrivant leur construction
- Exécution Lazy

Exemple

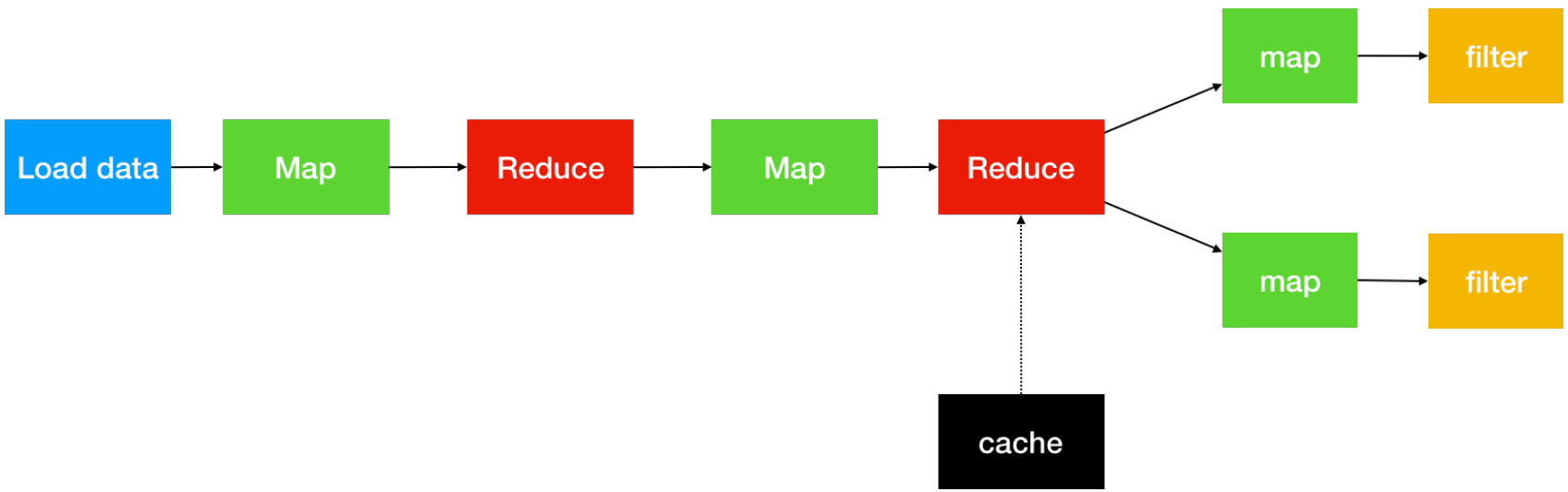
```
# sc is the sparkContext
# text_file_rdd only says that we will need to load a file
text_file_rdd = sc.textFile('mydataset.txt')


def my_map_function(line):
        print('Map executed...')
        return line.split(',')
# rdd only says that we will need to apply the map function
rdd = text_file_rdd.map(my_map_function)

rdd.take(15)
rdd.take(15)
```

Figure – `my_code.py`

Supposons que `mydataset.txt` contienne 10 éléments, combien de fois “Map executed...” sera affiché ?

Mise en cache

- Lorsqu'une partie de l'exécution est répétée, il est possible de la mettre en cache
- rdd.cache(),
- rdd. persist (MEMORY_ONLY | MEMORY_AND_DISK | ?).



Échantillonage

La première stratégie que l'on peut adopter face aux grands volumes de données est l'échantillonnage.

- Échantillonnage simple (avec ou sans remise) : Chaque donnée à la même probabilité d ?être tirée : $p = n/N$ dans le cas sans remise où n est la taille de notre échantillon, N, celle du jeu de données et on a généralement $n << N$

```
rdd.sample(False, 0.5) # sample(with_replacement, fraction)
```

- Échantillonnage stratifié (avec ou sans remise) : Ici, on suppose que chaque donnée est associée à une classe (ou plutôt une clé) et on souhaite que ces dernières restent représentées dans notre échantillon

```
rdd.sampleByKey(False, fraction) # où fraction est une map
# qui indique pour chaque clé (ou classe) la proportion à avoir
rdd.sampleByKeyExact(False, fraction)
```

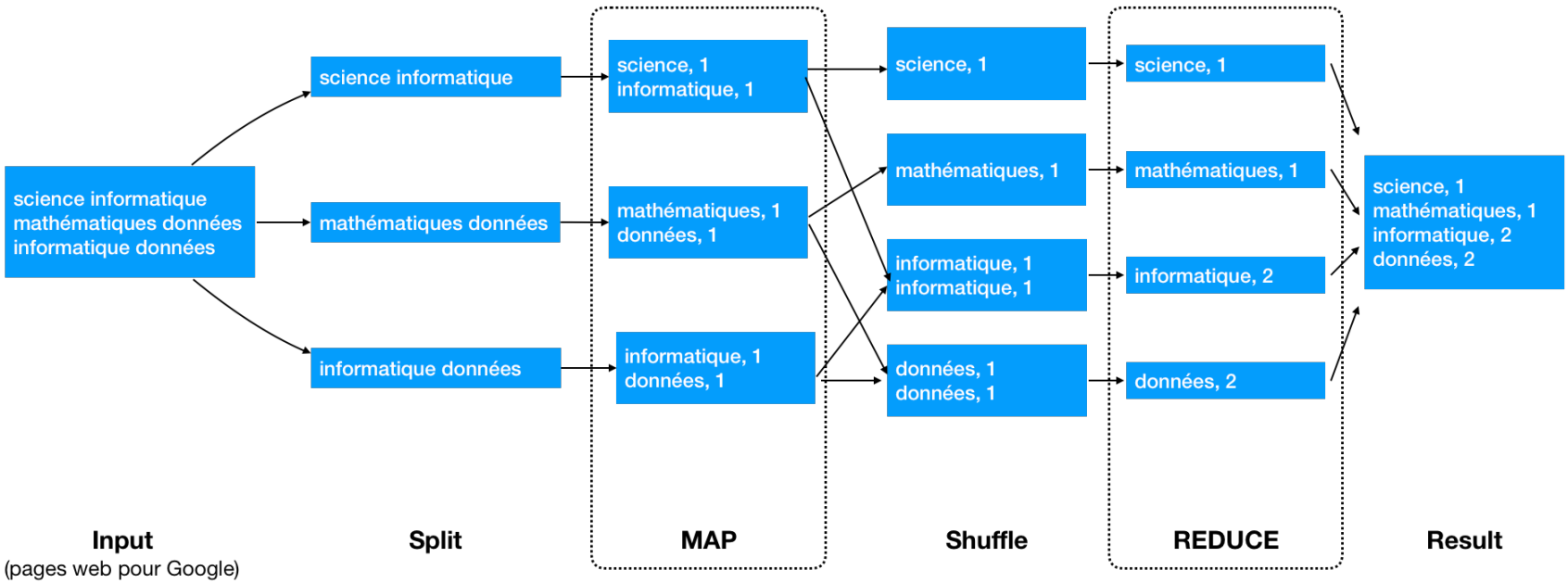
Réduction de la dimension

- *features selection*,
- ACP,
- *Deep learning*.

Map reduce 1/3

- L'ensemble de données à traiter est découpé en fragments (chunks)
- Chaque tâche Map est assignée à un noeud de calcul qui reçoit un ou plusieurs fragments que la tâche Map transforme en une séquence de paires (clé, valeur)
- Chaque tâche Reduce est associée à une ou plusieurs clés et est assignée à un njud de calcul
- Les paires (clé, valeur) produites par les Map sont groupées par clés et stockées sur les noeuds de calcul qui exécuteront les tâches Reduce respectives (étape shuffle)
- Chaque tâche Reduce combine, pour chaque clé qui lui est associée, les valeurs des paires (clé, valeur) avec cette clé

Map reduce (words count) 2/3



Map reduce 3/3

science
informatique
mathématiques
données
informatique
données

Figure – notre RDD `data`

```
data.map(lambda mot: (mot, 1))\
    .reduceByKey(lambda v1, v2: v1 + v2)
```

```
data.map(lambda mot: (clé, valeur))\
    .reduceByKey(
    lambda val_1, val_2: val_1 + val_2
)
```

Plan

Les *dataframes*

- Une table avec des colonnes auxquelles on peut associer un nom
- Immutable et Lazy
- Opérations SQL et SQL-like (where, group by, etc.)
- Similaire à R et Pandas

Les *dataframes* : exemples 1/3

1981,Mitterand
1988,Mitterand
1995,Chirac
2002,Chirac
2007,Sarkozy
2012,Hollande
2017,Macron

Figure – `dataset.csv`

```
df = spark.read.csv('dataset.txt')
df.printSchema()
df.withColumnRenamed("_c0", "year")
df.withColumnRenamed("_c1", "name")
df.printSchema()
# df.rdd enables to access the RDD
```

Figure – Sortie

Les *dataframes* : exemples 1/3

1981,Mitterand
1988,Mitterand
1995,Chirac
2002,Chirac
2007,Sarkozy
2012,Hollande
2017,Macron

Figure – `dataset.csv`

```
df = spark.read.csv('dataset.txt')
df.printSchema()
df.withColumnRenamed("_c0", "year")
df.withColumnRenamed("_c1", "name")
df.printSchema()
# df.rdd enables to access the RDD
```

Figure – Sortie

```
root
|– _c0 : string (nullable = true)
|– _c1 : string (nullable = true)
```

Les *dataframes* : exemples 1/3

1981,Mitterand
1988,Mitterand
1995,Chirac
2002,Chirac
2007,Sarkozy
2012,Hollande
2017,Macron

Figure – `dataset.csv`

```
df = spark.read.csv('dataset.txt')
df.printSchema()
df.withColumnRenamed("_c0", "year")
df.withColumnRenamed("_c1", "name")
df.printSchema()
# df.rdd enables to access the RDD
```

Figure – Sortie

```
root
|– _c0 : string (nullable = true)
|– _c1 : string (nullable = true)
root
|– _c0 : string (nullable = true)
|– _c1 : string (nullable = true)
```

Les *dataframes* : exemples 2/3

1981,Mitterand
1988,Mitterand
1995,Chirac
2002,Chirac
2007,Sarkozy
2012,Hollande
2017,Macron

Figure – `dataset.csv`

```
df = spark.read.csv('dataset.txt')
df.printSchema()
df = df.withColumnRenamed("_c0", "year")
df = df.withColumnRenamed("_c1", "name")
df = df.withColumn('year', df['year'].cast('int'
df.printSchema()
# df.rdd enables to access the RDD
```

Figure – Sortie

Les *dataframes* : exemples 2/3

1981,Mitterand
1988,Mitterand
1995,Chirac
2002,Chirac
2007,Sarkozy
2012,Hollande
2017,Macron

Figure – `dataset.csv`

```
df = spark.read.csv('dataset.txt')
df.printSchema()
df = df.withColumnRenamed("_c0", "year")
df = df.withColumnRenamed("_c1", "name")
df = df.withColumn('year', df['year'].cast('int'
df.printSchema()
# df.rdd enables to access the RDD
```

Figure – Sortie

```
root
|– _c0 : string (nullable = true)
|– _c1 : string (nullable = true)
```

Les *dataframes* : exemples 2/3

1981,Mitterand
1988,Mitterand
1995,Chirac
2002,Chirac
2007,Sarkozy
2012,Hollande
2017,Macron

Figure – `dataset.csv`

```
df = spark.read.csv('dataset.txt')
df.printSchema()
df = df.withColumnRenamed("_c0", "year")
df = df.withColumnRenamed("_c1", "name")
df = df.withColumn('year', df['year'].cast('int'
df.printSchema()
# df.rdd enables to access the RDD
```

Figure – Sortie

```
root
|– _c0 : string (nullable = true)
|– _c1 : string (nullable = true)
root
|– year : integer (nullable = true)
|– name : string (nullable = true)
```

Les *dataframes* : exemples 3/3

```
# how many times each president was elected?
df.groupBy('name').count()
df.show(5)
```

Les *dataframes* : exemples 3/3

```
# how many times each president was elected?
df.groupBy('name').count()
df.show(5)
```

```
+----+---------+
|year|     name|
+----+---------+
|1981|Mitterand|
|1988|Mitterand|
|1995|   Chirac|
|2002|   Chirac|
|2007|  Sarkozy|
+----+---------+
```

Les *dataframes* : exemples 3/3

```
# how many times each president was elected?
df = df.groupBy('name').count()
df.show(5)
```

```
+---------+-----+
|     name|count|
+---------+-----+
| Hollande|    1|
|   Chirac|    2|
|   Macron|    1|
|Mitterand|    2|
|  Sarkozy|    1|
+---------+-----+
```

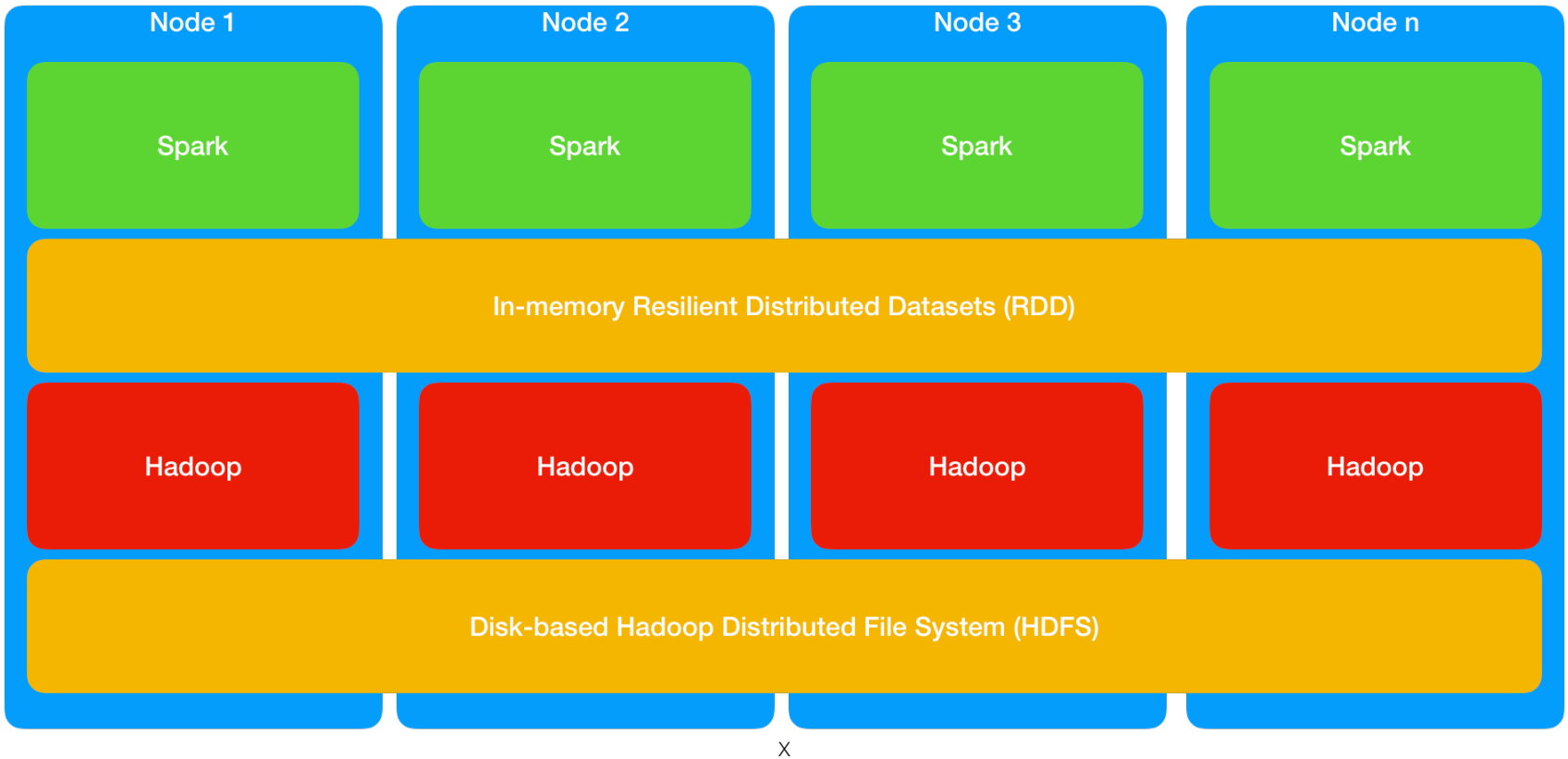
Plan

Introduction

Hadoop Distributed File System est un système de fichier et se structure autour des composants suivants :

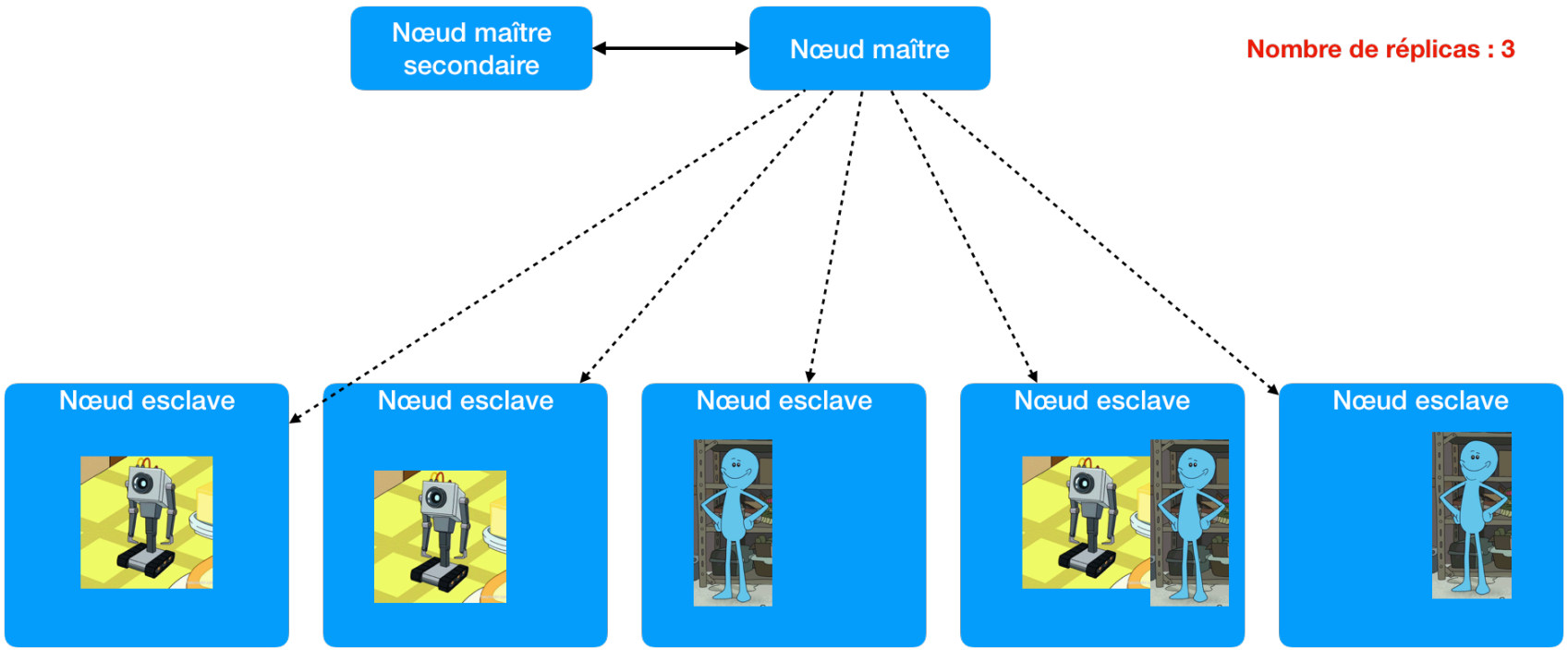
- **Name Node** : c'est le nœud maître qui relie les fichiers à leurs lieux de stockages (distribués, incluant l'arborescence, etc.), le client est directement connecté à ce nœud,
- **Data node** : il s'agit des nœuds qui s'occupent du stockage des blocs de donnée. Chaque bloc est répliqué sur plusieurs nœuds. Le *data node* avec la plus grosse bande passante disponible restitue le bloc lorsqu'il est demandé.

Architecture



X

Exemple


Nœud maître secondaire
Nœud maître
Nombre de réplicas : 3
Nœud esclave
Nœud esclave
Nœud esclave
Nœud esclave
Nœud esclave

Plan

Introduction

Pour des raisons historiques, Spark possède deux librairies de *machine learning* :

Introduction

Pour des raisons historiques, Spark possède deux librairies de *machine learning* :

1. **MLlib** – il s'agit de la plus ancienne
    - Construite sur les RDDs,

Introduction

Pour des raisons historiques, Spark possède deux librairies de *machine learning* :

1. **MLlib** – il s'agit de la plus ancienne
   - Construite sur les RDDs,
2. **ml** – librairie plus récente
   - Construite sur les Dataframe et/ou Dataset
   - Permet la construction d'un pipeline, etc.

Introduction

Pour des raisons historiques, Spark possède deux librairies de *machine learning* :

1. **MLlib** – il s'agit de la plus ancienne
   - Construite sur les RDDs,
2. **ml** – librairie plus récente
   - Construite sur les Dataframe et/ou Dataset
   - Permet la construction d'un pipeline, etc.

Préférer `ml` lorsque ce que l'on peut. `MLlib` permet d'exécuter des modèles qui n'ont pas encore été déplacés dans `ml`.

La suite de cette partie est construite autour d'exemples d'utilisation.

Système de recommandation

Une première stratégie : LSH

- Tous les utilisateurs notent des films et ces notes forment le profil de l ?utilisateur ainsi que celui de chaque film
- On peut vouloir chercher, étant donné un utilisateur donné, les utilisateurs les plus proches pour voir les notes qu ?ils ont mis et en déduire celle que mettrait l ?utilisateur courant
- Où appliquer ce raisonnement du point de vue des films

Une première stratégie : LSH

- Tous les utilisateurs notent des films et ces notes forment le profil de l ?utilisateur ainsi que celui de chaque film
- On peut vouloir chercher, étant donné un utilisateur donné, les utilisateurs les plus proches pour voir les notes qu ?ils ont mis et en déduire celle que mettrait l ?utilisateur courant
- Où appliquer ce raisonnement du point de vue des films

Comment calculer efficacement le voisinage d'un film ou d'un utilisateur ?

Définition d'une similarité

La similarité cosinus :

- Les données sont représentées par un vecteur numérique (pour des données textuelles, chaque dimension peut être la fréquence d ?utilisation d ?un mot)
- La similarité entre deux données est le cosinus de l ?angle que forment les deux vecteurs (0 si orthogonal 1 si angle de 0)
- Le cosinus :

$$\text{similarity}(u, v) = cos(\theta_{u,v}) = \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \sqrt{\sum_{i=1}^{n} u_i^2}}$$

Définition d'une similarité

La similarité cosinus :

- Les données sont représentées par un vecteur numérique (pour des données textuelles, chaque dimension peut être la fréquence d ?utilisation d ?un mot)
- La similarité entre deux données est le cosinus de l ?angle que forment les deux vecteurs (0 si orthogonal 1 si angle de 0)
- Le cosinus :

$$\text{similarity}(u, v) = cos(\theta_{u,v}) = \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \sqrt{\sum_{i=1}^{n} u_i^2}}$$

Comment calculer efficacement un voisinage ?

Locality Sensitive Hashing (LSH)

- Soit H_{cos} un ensemble de fonction de hachage défini de la manière suivante :
  - soit $h_v \in H_{cos}$, alors

$$h_v(x) = \begin{cases} 1 \text{ si } \langle x, v \rangle \geq 0 \\ 0 \text{ si } \langle x, v \rangle < 0 \end{cases}$$

  - Deux données formant un angle de 0 ne peuvent pas être séparées par h_v et deux données formant un angle de $\pi/2$ ne peuvent pas ne pas l'être.
- Le hach attribué à une donné x est le vecteur $[h_v(x)]_{h_v \in H_{cos}}$.
- Cet algorithme nous permet de recherches des vecteurs proches efficacement.

La factorisation de matrice pour la recommandation

	Film 1	Série 1	Film 2	Livre 1
Alice	4	?	4	3
Bob	4	?	3	?
Rick	?	5	4	?
Summer	?	5	?	4

La factorisation de matrice pour la recommandation

				Film 1	Série 1	Film 2	Livre 1
				Θ_{11}	Θ_{21}	Θ_{31}	Θ_{41}
				Θ_{12}	Θ_{22}	Θ_{32}	Θ_{42}
				Θ_{13}	Θ_{23}	Θ_{33}	Θ_{43}
Alice	-0.62	-0.34	-0.60	4	?	4	3
Bob	-0.2	-0.42	-0.77	4	?	3	?
Rick	-0.58	-1.23	-0.02	?	5	4	?
Summer	-0.97	-0.41	-0.48	?	5	?	4

La factorisation de matrice pour la recommandation

				Film 1	Série 1	Film 2	Livre 1
				-1.83	-1.57	-3.4	-2.9
				-2.1	-3.33	-1.6	-1.2
				-3.6	-0.07	-2.2	-1.4
Alice	Θ_{11}	Θ_{12}	Θ_{13}	4	?	4	3
Bob	Θ_{21}	Θ_{22}	Θ_{23}	4	?	3	?
Rick	Θ_{31}	Θ_{32}	Θ_{33}	?	5	4	?
Summer	Θ_{41}	Θ_{42}	Θ_{43}	?	5	?	4

La factorisation de matrice pour la recommandation

- La matrice étant incomplète, les stratégies classiques ne fonctionnent pas,
- Minimisation de l'erreur :

$$J(U,V) = \sum_{i=1}^{M}\sum_{j=1}^{N}(r_{ij} - \hat{r_{ij}})^2 + \lambda R(U,V)$$
$$= \sum_{i=1}^{M}\sum_{j=1}^{N}(r_{ij} - \langle U_i, V_j \rangle)^2 + \lambda(\|U\|^2 + \|V\|^2)$$

- L'algorithme des moindres carrés alternés est utilisé (on minimise un coup l'erreur pour les utilisateurs en considérant les objets constant puis inversement) : ALS.

La factorisation de matrice pour la recommandation

```
from pyspark.ml.recommendation import ALS

als = ALS(userCol="user_id", itemCol="item_id", ratingCol="rating")
model = als.fit(training)

# prediction
predictions = model.transform(test)
```

Détection de SPAM

Pipeline

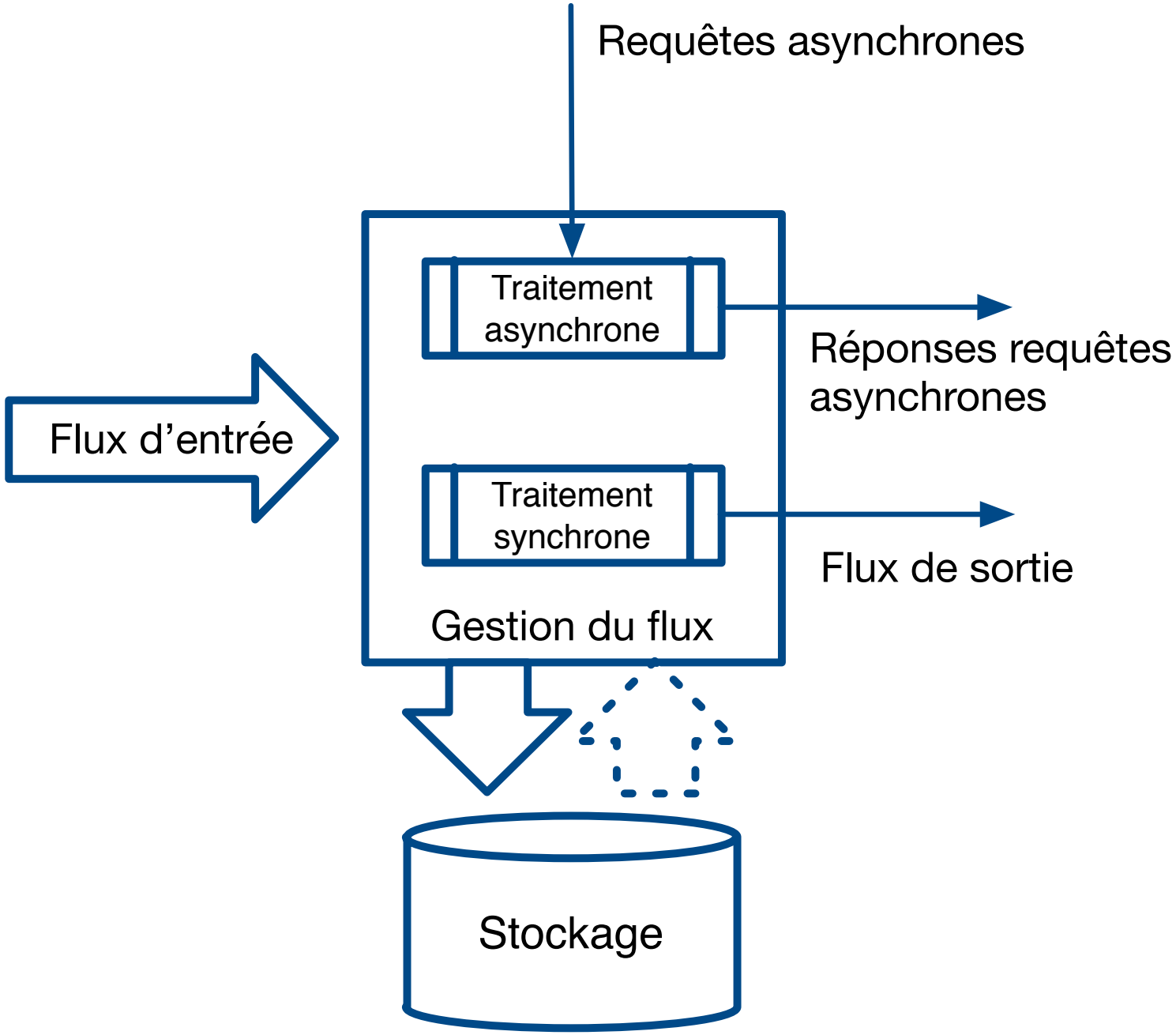
SPAM	SMS
0	Nah I don't think he goes to usf, he lives around here though
1	WINNER!! As a valued network customer you have been selected to receivea ?900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.

La factorisation de matrice pour la recommandation

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

tokenizer = Tokenizer(inputCol="sms", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)

pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# train
model = pipeline.fit(train)

# prediction
predictions = model.transform(test)
```

Gestion de flux

Flux de données (Vélocité)



Échantillonnage

- Lorsqu'il y a impossibilité d'analyser les données de manière exhaustive
- Sélectionner chacune des données avec une probabilité p
  - Capteurs netflows sur les routeursă : seule une partie des paquets sont analysés
- Sélectionner chacune des données suivant la valeur de certains de leurs attributs
  - Selon une sélection prédéfinie
  - Car la valeur n'a pas encore été observée

Un exemple avec le filtre de *Bloom*

- Imaginons un capteur sur le réseau qui log les paquets qui le traversent
- On souhaite ne garder que les paquets dont l'IP source appartient (n'appartient pas) à une liste de 100000 IPs. Comment faire ?

Le filtre de *Bloom*

- On dispose d'un tableau $\mathcal{T}$ de booléens de taille m (ici 10) :

0	0	0	0	0	0	0	0	0	0

- k fonction de hachages $h_i : \mathcal{X} \mapsto [1, m]$ où $\mathcal{X}$ est l'ensemble des données (e.g. adresses IPs) : e.g. $h_i('192.168.0.1') = 8$,
- Ajouter un élément à notre filtre consiste à passer les booléens $\forall i, \mathcal{T}[h_i(d)] = 1$.

Un exemple 1/5

- Exemple : Soit 2 fonctions de hachage :

$$h_1('192.168.0.1') = 6, h_2('192.168.0.1') = 4$$

0	0	0	0	0	0	0	0	0	0

Un exemple 2/5

- Exemple : Soit 2 fonctions de hachage :

$$h_1('192.168.0.1') = 6, h_2('192.168.0.1') = 4$$

0	0	0	1	0	1	0	0	0	0

Un exemple 3/5

- Exemple : Soit 2 fonctions de hachage :

$$h_1('192.168.0.2') = 2, h_2('192.168.0.2') = 6$$

0	0	0	1	0	1	0	0	0	0

Un exemple 4/5

- Exemple : Soit 2 fonctions de hachage :

$$h_1('192.168.0.2') = 2, h_2('192.168.0.2') = 6$$

0	1	0	1	0	1	0	0	0	0

Un exemple 5/5

- On teste la présence de l'adresse suivante :

$$h_1('192.168.0.3') = 4, h_2('192.168.0.3') = 2$$

0	1	0	1	0	1	0	0	0	0

Probabilité de faux négatifs

- Quelle est la probabilité d'obtenir un faux négatif ?

Probabilité de faux négatifs

- Quelle est la probabilité d'obtenir un faux négatif ?
- 0 !

Probabilité de faux positifs

Probabilité de faux positifs

- Supposons que chaque fonction de hachage a une probabilité uniforme sur $[1, m]$. La probabilité qu'un bit du tableau devienne positif suite à l'application d'une fonction de hachage est donc $1/m$ et $1 - 1/m$ qu'il reste négatif,

Probabilité de faux positifs

- Supposons que chaque fonction de hachage a une probabilité uniforme sur $[1, m]$. La probabilité qu'un bit du tableau devienne positif suite à l'application d'une fonction de hachage est donc $1/m$ et $1 - 1/m$ qu'il reste négatif,
- Après l'application de k fonctions de hachage on obtient que la probabilité qu'un bit reste négatif est $(1 - 1/m)^k$.

Probabilité de faux positifs

- Supposons que chaque fonction de hachage a une probabilité uniforme sur $[1, m]$. La probabilité qu'un bit du tableau devienne positif suite à l'application d'une fonction de hachage est donc $1/m$ et $1 - 1/m$ qu'il reste négatif,
- Après l'application de k fonctions de hachage on obtient que la probabilité qu'un bit reste négatif est $(1 - 1/m)^k$.
- Après avoir ajouté n éléments, on a $(1 - 1/m)^{kn}$,

Probabilité de faux positifs

- Supposons que chaque fonction de hachage a une probabilité uniforme sur $[1, m]$. La probabilité qu'un bit du tableau devienne positif suite à l'application d'une fonction de hachage est donc $1/m$ et $1 - 1/m$ qu'il reste négatif,
- Après l'application de k fonctions de hachage on obtient que la probabilité qu'un bit reste négatif est $(1 - 1/m)^k$.
- Après avoir ajouté n éléments, on a $(1 - 1/m)^{kn}$,
- La probabilité que ce bit reste à 1 est $1 - (1 - 1/m)^{kn}$.

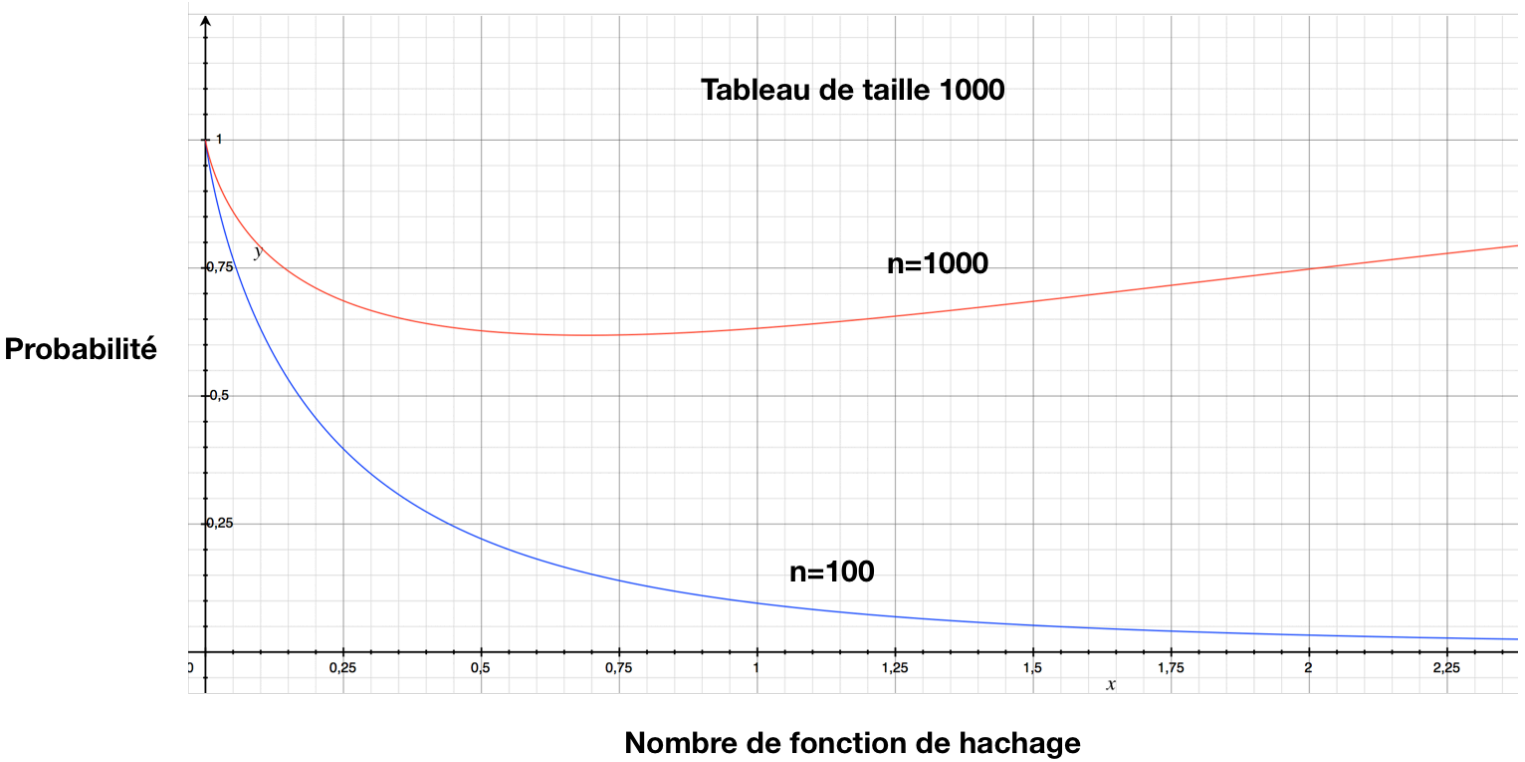
Probabilité de faux positifs

- Supposons que chaque fonction de hachage a une probabilité uniforme sur $[1, m]$. La probabilité qu'un bit du tableau devienne positif suite à l'application d'une fonction de hachage est donc $1/m$ et $1 - 1/m$ qu'il reste négatif,
- Après l'application de k fonctions de hachage on obtient que la probabilité qu'un bit reste négatif est $(1 - 1/m)^k$.
- Après avoir ajouté n éléments, on a $(1 - 1/m)^{kn}$,
- La probabilité que ce bit reste à 1 est $1 - (1 - 1/m)^{kn}$.
- la probabilité d'un faux positif est que les k bits de ses fonctions de hachage soient déjà à 1 : $(1 - (1 - 1/m)^{kn})^k$.

Probabilité de faux positifs

- Supposons que chaque fonction de hachage a une probabilité uniforme sur $[1, m]$. La probabilité qu'un bit du tableau devienne positif suite à l'application d'une fonction de hachage est donc $1/m$ et $1 - 1/m$ qu'il reste négatif,
- Après l'application de k fonctions de hachage on obtient que la probabilité qu'un bit reste négatif est $(1 - 1/m)^k$.
- Après avoir ajouté n éléments, on a $(1 - 1/m)^{kn}$,
- La probabilité que ce bit reste à 1 est $1 - (1 - 1/m)^{kn}$.
- la probabilité d'un faux positif est que les k bits de ses fonctions de hachage soient déjà à 1 : $(1 - (1 - 1/m)^{kn})^k$.
- Dans notre exemple, on a : $(1 - (1 - 1/10)^{2\times 2})^2 = 0.11$

Filtre de *Bloom*



Fenêtre temporelle

- Même après échantillonnage, le coût peut rester excessif,
- L'historique lointain n'est pas nécessairement pertinent.

Fenêtre temporelle

- Même après échantillonnage, le coût peut rester excessif,
- L'historique lointain n'est pas nécessairement pertinent.
- On peut se limiter au traitement des données à l'intérieur d'une fenêtre glissante

- On appelle les cases bleues des "tranches"

Fenêtre temporelle

- Même après échantillonnage, le coût peut rester excessif,
- L'historique lointain n'est pas nécessairement pertinent.
- On peut se limiter au traitement des données à l'intérieur d'une fenêtre glissante

- On appelle les cases bleues des "tranches"
  - On peut pondérer l'importance d'une tranche par son ancienneté dans la fenêtre.

Fenêtre temporelle

- Même après échantillonnage, le coût peut rester excessif,
- L'historique lointain n'est pas nécessairement pertinent.
- On peut se limiter au traitement des données à l'intérieur d'une fenêtre glissante

- On appelle les cases bleues des "tranches"
  - On peut pondérer l'importance d'une tranche par son ancienneté dans la fenêtre.
- Les fenêtres peuvent être de longueur fixe ou de volume fixe.

Avec Spark

Exemple de code :

```
ssc = StreamingContext(sc, 1) # sparkContext, delay
stream = ssc.textFileStream('path/to/folder')
stream.foreachRDD(my_function_on_RDD)
ssc.start() # prend la main dans jupyter...
# ssc.stop(True, True)
```

Les fenêtres temporelles dans Spark

- `DStream.window(windowLength, slideInterval)`
- `DStream.countByWindow(windowLength, slideInterval)`
- `DStream.reduceByWindow(func, windowLength, slideInterval)`
- `DStream.reduceByKeyAndWindow(func, windowLength, slideInterval)`