

# Bases de données NoSQL

## Chapitre 2 : Fondements des BD NoSQL

Pr. ZAIDOUNI Dounia

Filière: ASEDS, S2, INE1  
Institut National des Postes et Télécommunications (INPT)

29 Avril 2025

# Présentation du syllabus du cours

- Chapitre 1: Introduction aux BD NoSQL
- Chapitre 2: Fondements des BD NoSQL :
  - La réPLICATION
  - Les techniques de partionnement
  - Le modèle programmation MapReduce
- Chapitre 3: Typologie des BD NoSQL
- Chapitre 4: MongoDB
- Chapitre 5: Apache Hbase

# Outline

## 1 Fondements des systèmes NoSQL

# Plan

## 1 Fondements des systèmes NoSQL

- La réPLICATION

- Pourquoi la réPLICATION
- Niveau de réPLICATION
- Mécanisme de réPLICATION

- Les techniques de partionnement

- Partitionnement simple (Sharding)
- Hachage cohérent (Consistent hashing)

- Modèle de programmation MapReduce

# Pourquoi la réPLICATION

- La solution traditionnelle pour tolérer les pannes consiste à effectuer des sauvegardes régulières (Checkpoints). La réPLICATION est une autre manière pour tolérer les pannes, c'est une sorte de sauvegarde continue.
- Les systèmes NoSQL utilisent la réPLICATION pour assurer:
  - **Disponibilité:** La réPLICATION permet d'assurer la disponibilité constante du système (en cas de panne, la tâche est prise en charge par l'autre composant).
  - **Scalabilité (lecture):** Si une donnée est disponible sur plusieurs machines, il devient possible de distribuer les requêtes (en lecture) sur ces machines (Par exemple: les applications web).
  - **Scalabilité (écriture):** Enfin, on peut penser à distribuer aussi les requêtes en écriture, mais là on se retrouve face à des problèmes d'écritures concurrentes et de réconciliation.

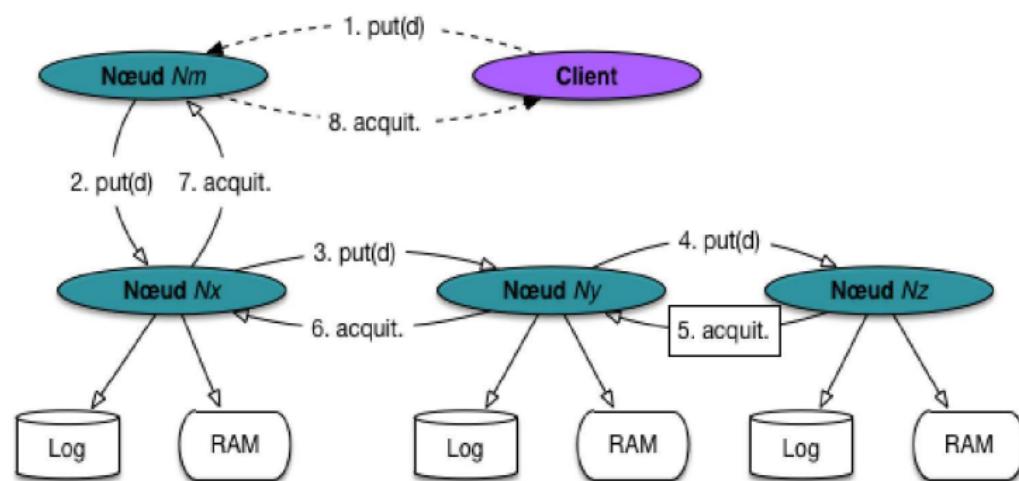
# Niveau de réPLICATION

- Il dépend notamment du budget qu'on est prêt à allouer à la sécurité des données.
- On peut considérer que 3 copies constituent un bon niveau de sécurité.
- Par exemple :
  - Une stratégie possible est de placer un document sur un serveur dans un rack, une copie dans un autre rack pour qu'elle reste accessible en cas de coupure réseau, et une troisième dans un autre centre de données pour assurer la survie d'au moins une copie en cas d'accident grave.

# Mécanisme de réPLICATION

- L'application client demande au système l'écriture d'un document  $d$ . Cela signifie qu'il existe un des noeuds du système, disons **N<sub>m</sub>**, qui constitue l'interlocuteur du client.
- Dans une architecture Maître-Esclave, N<sub>m</sub> est typiquement le Maître.
- N<sub>m</sub> va identifier un noeud responsable du stockage de  $d$ , disons **N<sub>x</sub>**. Le processus de réPLICATION fonctionne alors comme suit:
  - N<sub>x</sub> écrit localement le document  $d$ ;
  - N<sub>x</sub> transmet la demande d'écriture à un ou plusieurs autres serveurs, N<sub>y</sub>, N<sub>z</sub>, qui à leur tour effectuent l'écriture.
  - N<sub>x</sub>, N<sub>y</sub>, N<sub>z</sub> renvoient un acquittement à N<sub>m</sub> confirmant l'écriture.
  - N<sub>m</sub> renvoie un acquittement au client pour lui confirmer que  $d$  a bien été enregistré.
- Il existe bien sûr d'autres variantes.
  - Par exemple: N<sub>m</sub> peut se charger de distribuer les trois requêtes d'écriture, au lieu de créer une chaîne de réPLICATION, etc.

# RéPLICATION avec écritures synchrones



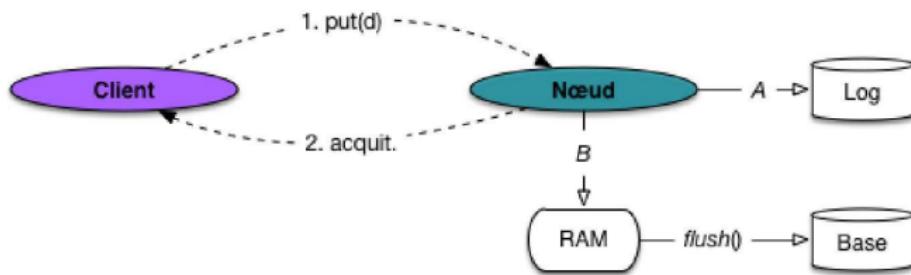
# RéPLICATION avec écritures synchrones

- Ce type de réPLICATION est utilisé par les SGBD qui exigent la durabilité.
- L'acquittement n'est donné au client que quand la donnée est vraiment enregistrée de manière permanente. Dans la figure précédente :
  - Quand le client reçoit l'acquittement, il est sûr que trois copies de d sont effectivement enregistrées de manière durable dans le système.
  - Cela nécessite une chaîne comprenant 8 messages, tout obstacle le long du chemin (un serveur temporairement surchargé par exemple) risquant d'allonger considérablement le temps d'attente.
- Entre la demande d'écriture et la réception de l'acquittement, le client attend, ce qui bloque l'application client, prix à payer pour la sécurité.
- Dans ce cas, le fait d'effectuer plusieurs copies allonge encore le temps d'attente de l'application.

# Techniques pour limiter le temps d'attente

- Deux techniques sont utilisées pour limiter le temps d'attente, toutes les deux affectant (un peu) la sécurité des opérations :
  - Ecriture en mémoire RAM, et fichier journal (log);
  - RéPLICATION asynchrone.

# Ecriture en mémoire RAM et fichier journal



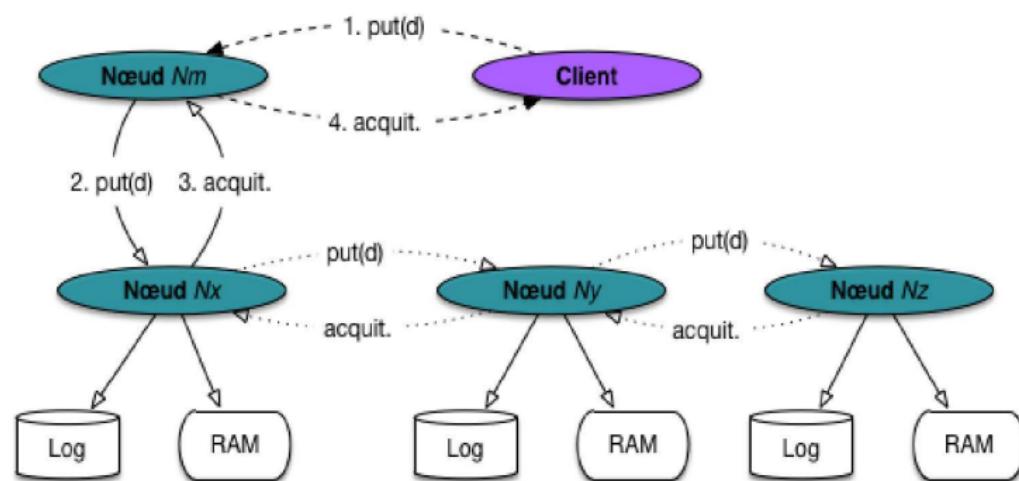
## Ecriture en mémoire RAM et fichier journal

- Cette technique est très classique et utilisée par tous les SGBD.
- Au lieu d'effectuer des écritures répétées sur le disque sans ordre pré-défini qui imposent à chaque fois un déplacement de la tête de lecture et donc une latence de quelques ms, on écrit séquentiellement dans un fichier de journalisation (log) et on place également la donnée en mémoire RAM.
- À terme, le contenu de la mémoire RAM, marqué comme contenant des données modifiées, sera écrit sur le disque dans les fichiers de la base de données (**opération de flush()**).
  - Cela permet de grouper les opérations d'écritures et donc de revenir à des entrées/sorties séquentielles sur le disque, aussi bien dans le fichier journal que dans la base principale.

## Ecriture en mémoire RAM et fichier journal

- En cas de panne avant l'opération de flush(), les données modifiées n'ont pas été écrites dans la base, mais le journal (log) est préservé.
- La reprise sur panne consiste à ré-effectuer les opérations enregistrées dans le log.
- Le fichier log et la base devraient être sur des disques distincts.

# RéPLICATION avec écritures asynchrones



# RéPLICATION avec écritures asynchrones

- Pour limiter le temps d'écriture, cette technique est le recours à des écritures asynchrones.
- Contrairement à la réPLICATION avec écritures synchrones :
  - Le serveur Nx qui reçoit la requête va effectuer l'écriture et envoyer immédiatement l'acquittement au client, lui rendant ainsi la main et permettant la poursuite de son exécution.
  - Après l'acquittement, Nx commence l'envoi des messages pour la réPLICATION en mode asynchrone.

# RéPLICATION avec écritures asynchrones

- Dans ce scénario de RéPLICATION avec écritures asynchrones, beaucoup plus rapide pour le client, deux phénomènes apparaissent :
  - Le client reçoit un acquittement alors que la réPLICATION n'est pas complète; il n'y a donc pas à ce stade de garantie complète de sécurité;
  - Le client poursuit son exécution alors que toutes les copies de d ne sont pas encore mises à jour; il se peut alors qu'une lecture renvoie une des copies obsolètes de d.
- Il y a donc un risque pour la cohérence des données. C'est un problème sérieux, caractéristique des systèmes distribués en général, du NoSQL en particulier.

# Partitionnement simple (Sharding)

- Supposant qu'on a un cluster de  $n$  noeuds serveurs notés  $\{N_0, \dots, N_{n-1}\}$  et qu'on veut stocker des données sous forme de paires (Key, Values) dans ces différents noeuds.
- Le partitionnement simple consiste à partitionner de manière horizontale les données sous forme clés-valeurs et de les stocker dans les noeuds serveurs en utilisant une fonction de hachage. Par exemple:

$$\text{Identifiant}_{\text{noeud}} = \text{HashFunc}(\text{Key}) \bmod n$$

- Il faut bien choisir la fonction de hachage afin d'éviter les collisions et afin de garantir une répartition uniforme des données sur le serveurs.

# Problème du partitionnement simple

- Le partitionnement simple entraîne des migrations des données lorsque des nœuds sont **ajoutés** ou **retirés** du cluster.
- Exemple : La table ci-dessous illustre cette problématique de migration des données avec deux clusters, un composé initialement de 4 noeuds puis de 5 noeuds.

Le résultat du modulo indique l'identité du noeud  $N_i$ ,  $i=0..4$ , où la donnée sera stockée.

Clé	Cluster 4 nœuds	Cluster 5 nœuds	Migration des données
45	$45 \bmod 4 = 1 \equiv N1$	$45 \bmod 5 = 0 \equiv N0$	$N1 \rightarrow N0$
46	$46 \bmod 4 = 2 \equiv N2$	$46 \bmod 5 = 1 \equiv N1$	$N2 \rightarrow N1$
47	$47 \bmod 4 = 3 \equiv N3$	$47 \bmod 5 = 2 \equiv N2$	$N3 \rightarrow N2$
48	$48 \bmod 4 = 0 \equiv N0$	$48 \bmod 5 = 3 \equiv N3$	$N0 \rightarrow N3$
49	$49 \bmod 4 = 1 \equiv N1$	$49 \bmod 5 = 4 \equiv N4$	$N1 \rightarrow N4$
50	$50 \bmod 4 = 2 \equiv N2$	$50 \bmod 5 = 0 \equiv N0$	$N2 \rightarrow N0$

# Hachage cohérent (Consistent hashing)

- Afin d'éviter le problème de migration massive de données entre les noeuds lors de l'ajout ou du retrait de noeuds, un algorithme plus sophistiqué appelé 'hachage cohérent' est utilisé.
- Il est d'abord conçu et utilisé pour des caches distribués (memcached).
- Il est popularisé pour la gestion de données par le système Dynamo (Amazon, 2007).
- Maintenant intégré à de nombreux systèmes : Apache Cassandra, Voldemort, Riak,...

Nous allons expliquer le principe du hachage cohérent en se basant sur le fonctionnement du système NoSQL d'Apache Cassandra.

# Apache Cassandra

- Apache Cassandra a une architecture 'peer-to-peer' c-à-d. sans notion de serveur maître-esclave et utilisant une topologie en **anneau**.
- Dans ce type d'architecture en anneau chaque noeud joue le même rôle et ils communiquent entre eux grâce à un protocole nommé 'gossip protocol'. Ce protocole permet aux noeuds du cluster d'échanger des informations concernant leur état, l'ajout ou le retrait d'un noeud, la mise à jour des données.
- Concernant l'aspect de mise à l'échelle, elle est gérée en ajoutant, 'à chaud' (c-à-d. sans arrêt/redémarrage du cluster), de nouveaux nœuds.
- Dans ce système NoSQL les données doivent être partitionnées et distribuées équitablement entre les différents noeuds composant le cluster.

# Principe du hachage cohérent (1/4)

- L'idée de base du 'hachage cohérent' est d'appliquer une même fonction de hachage :

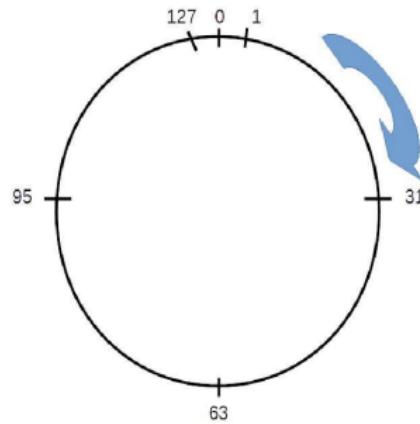
$$\text{HashFunc( Id or Key ) mod } 2^m$$

aux identifiants des noeuds du cluster (**Id** généralement égale à l'adresses IP combinées au port de communication) et aux clés primaires (**Key**) des lignes des tables de données afin de générer des valeurs dans le même espace d'identifiants.

- Ces valeurs de hachage sont ensuite placées sur un cercle.
- Supposons que les valeurs de hachage sont encodées sur **m** bits. Nous disposons alors de  $2^m$  points. Chaque identifiant de noeud ou de clé primaire peut alors être choisi parmi ces points.

## Principe du hachage cohérent (2/4)

- Si  $m=7$ , nous disposons de  $2^7 = 128$  points (entiers) compris entre 0 et 127. Si nous les plaçons sur un cercle nous avons la représentation suivante :



## Principe du hachage cohérent (3/4)

- Concernant le routage des messages dans une topologie en anneau, chaque noeud connaît ses successeurs (noeuds situés dans le sens des aiguilles d'une montre) et parfois ses prédecesseurs.
- Supposons que nous disposons de 4 noeuds où les valeurs de hachage, encodées sur  $m=7$  bits, sont indiquées dans la table suivante :

Clé	Identifiant sur le cercle
$N_0$	$HashFunc(N_0) \bmod 2^7 = 0$
$N_1$	$HashFunc(N_1) \bmod 2^7 = 31$
$N_2$	$HashFunc(N_2) \bmod 2^7 = 63$
$N_3$	$HashFunc(N_3) \bmod 2^7 = 95$

## Principe du hachage cohérent (4/4)

- L'**algorithme d'affectation** des identifiants des lignes de données aux noeuds d'un cluster est le suivant :
  - En se déplaçant dans le sens des aiguilles d'une montre :  
Chaque noeud est responsable des données dont les valeurs de hachage des clés primaires sont inférieures à son identifiant et supérieures à l'identifiant du noeud qui le précéde.
- Nous allons présenter un **exemple** avec un cluster composé de 4 noeuds ( $N_0, \dots, N_3$ ) puis nous analysons l'impact de l'**ajout** d'un cinquième noeud ( $N_4$ ).

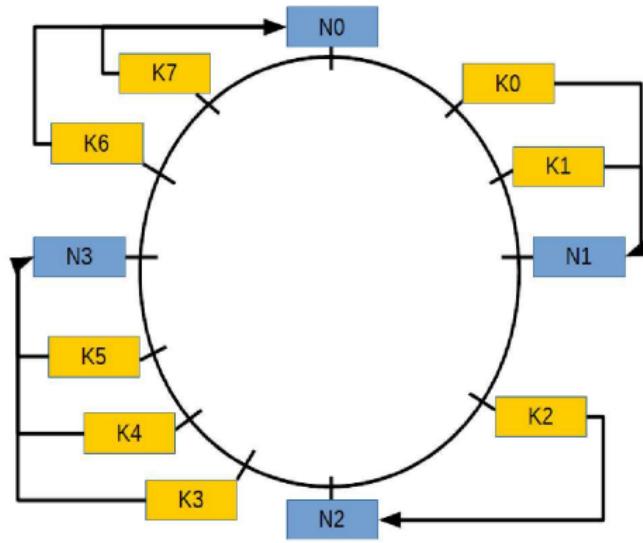
## Exemple (1/3)

- Nous posons :  $K_i$  = clé primaire de la ligne de donnée  $i$ .
- Les valeurs de hachage pour les clés primaires des lignes de données et les identifiants de noeuds sont regroupées dans la table suivante.

Clé	Identifiant sur le cercle
$N_0$	$HashFunc(N_0) \bmod 2^7 = 0$
$K_0$	$HashFunc(K_0) \bmod 2^7 = 20$
$K_1$	$HashFunc(K_1) \bmod 2^7 = 30$
$N_1$	$HashFunc(N_1) \bmod 2^7 = 31$
$K_2$	$HashFunc(K_2) \bmod 2^7 = 50$
$N_2$	$HashFunc(N_2) \bmod 2^7 = 63$
$K_3$	$HashFunc(K_3) \bmod 2^7 = 70$
$K_4$	$HashFunc(K_4) \bmod 2^7 = 85$
$K_5$	$HashFunc(K_5) \bmod 2^7 = 90$
$N_3$	$HashFunc(N_3) \bmod 2^7 = 95$
$K_6$	$HashFunc(K_6) \bmod 2^7 = 100$
$K_7$	$HashFunc(K_7) \bmod 2^7 = 110$
$N_4$	$HashFunc(N_4) \bmod 2^7 = 80$

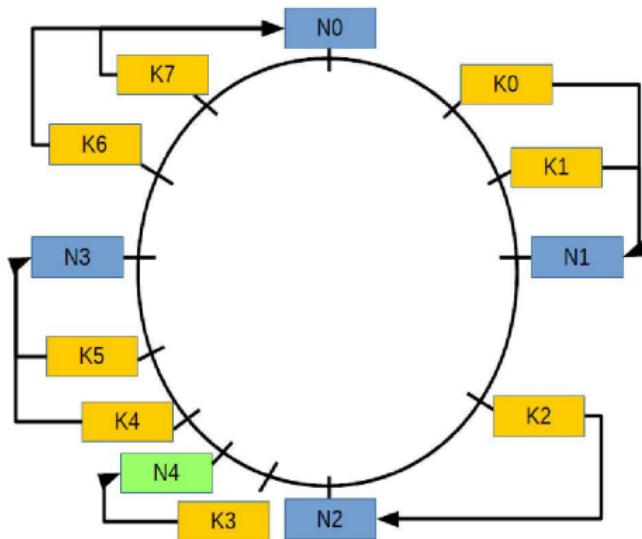
## Exemple (2/3)

- Si nous posons les identifiants des noeuds et des clés primaires sur un anneau et que nous utilisons l'algorithme d'affectation des identifiants des lignes de données. Nous obtenons la figure suivante :



## Exemple (3/3)

- Nous introduisons un nouveau noeud N4 d'ID 80.
- La nouvelle affectation des lignes de données (dans un cluster de 5 noeuds dans une topologie 'peer-to-peer' en anneau après ajout du noeud N4) est représentée dans la figure suivante :



# Modèle de programmation MapReduce

- MapReduce est un modèle de programmation dans lequel sont effectués des calculs parallèles, et souvent distribués, de données potentiellement très volumineuses.
- Inventé par Google pour le traitement de gros volumes de données en environnement distribué :
  - Il permet de répartir la charge sur un grand nombre de serveurs (cluster).
  - Il fait abstraction quasi-totale de l'infrastructure matérielle :
    - Il gère entièrement, de façon transparente le cluster, la distribution de données, la répartition de la charge, et la tolérance aux pannes.
  - Il permet d'ajouter des machines afin d'augmenter la performance (scalable friendly).

# Modèle de programmation MapReduce

- Le modèle MapReduce connaît un vif succès auprès de sociétés possédant d'importants centres de traitement de données telles Amazon ou Facebook.
- Ce modèle est aussi utilisé au sein des plateformes du Cloud computing.
- La librairie MapReduce existe dans plusieurs langages (C++, C#, Java, Python, Ruby, ...).
- De nombreux frameworks ont vu le jour afin d'implémenter le MapReduce. Le plus connu est 'Apache Hadoop'.



# Divers usages de MapReduce

- Utilisé par les grands acteurs du Web notamment pour : construire les index (Google Search), détection de spam (Yahoo), Data Mining (Facebook), ...
- Autres utilisations :
  - De l'analyse d'images astronomique, de la bio-informatique, de la simulation météorologique, des statistiques, etc.
  - Le calcul de la taille de plusieurs milliers de documents.
  - Trouver le nombre d'occurrences d'un pattern dans un très grand volume de données.
  - Classifier de très grands volumes de données provenant par exemple de paniers d'achats de clients (Data Mining).

# Fonctionnement de MapReduce

On distingue donc 5 étapes distinctes dans un traitement MapReduce:

- **Préparer (Pre-Process)** les données en entrée.
- **Découper (Split)** les données d'entrée en plusieurs fragments.
- **Mapper (Map)** chacun de ces fragments pour obtenir des couples (clé;valeur).
  - Elle prend en entrée un ensemble de "Clé,Valeurs" et retourne une liste intermédiaire de "Clé1,Valeur1":  $\text{Map}(\text{key,value}) \rightarrow \text{list}(\text{key1,value1})$
- **Grouper (Shuffle)** ces couples (clé;valeur) par clé.
- **Réduire (Reduce)** les groupes indexés par clé en une forme finale, avec une valeur pour chacune des clés distinctes.
  - Elle prend en entrée une liste intermédiaire de "Clé1,Valeur1" et fournit en sortie une ensemble de "Clé1,Valeur2":  
 $\text{Reduce}(\text{clé1,list(valeur1)}) \rightarrow \text{valeur2}$

# Exemple d'un programme MapReduce

- Imaginons qu'on a un texte écrit en langue française et qu'on souhaite déterminer pour un travail de recherche quels sont les mots les plus utilisés au sein de ce texte.
- Nous allons dérouler toutes les étapes du traitement MapReduce sur cet exemple :
  - Pre-process
  - Map
  - Split
  - Shuffle
  - Reduce

# Pre-Process

- Etape 'Pre-process':
  - Les données d'entrée sont sous forme de texte en français.

Excellence, Polyvalence et Ouverture sont les valeurs de l'INPT.  
L'excellence, un challenge.

Polyvalence et enrichissement des compétences.

Ouverture sur le monde.

Ouverture sur l'entreprise et ouverture sur l'autre.

# Pre-Process

- Etape 'Pre-process':

- Les données d'entrée sont sous forme de texte en français.

Excellence, Polyvalence et Ouverture sont les valeurs de l'INPT.  
L'excellence, un challenge.

Polyvalence et enrichissement des compétences.

Ouverture sur le monde.

Ouverture sur l'entreprise et ouverture sur l'autre.

- Pour simplifier les choses, avant le découpage, on va supprimer toute ponctuation et tous les caractères accentués et on va passer l'intégralité du texte en minuscules.

excellence polyvalence et ouverture sont les valeurs de l inpt  
l excellence un challenge  
polyvalence et enrichissement des competences  
ouverture sur le monde  
ouverture sur l entreprise et ouverture sur l autre

# Split

- Etape 'Split':

- On va déterminer une manière de découper les données d'entrée pour que chacune des machines puisse travailler sur une partie du texte.
- On peut par exemple décider de découper les données d'entrée ligne par ligne. Chacune des lignes du texte sera un fragment de nos données d'entrée.

# Split

- Etape 'Split':
  - On va déterminer une manière de découper les données d'entrée pour que chacune des machines puisse travailler sur une partie du texte.
  - On peut par exemple décider de découper les données d'entrée ligne par ligne. Chacune des lignes du texte sera un fragment de nos données d'entrée.
- Après découpage, on obtient 4 fragments depuis nos données d'entrée:

excellence polyvalence et ouverture sont les valeurs de l input

l excellence un challenge

Polyvalence et enrichissement des compétences

ouverture sur le monde

ouverture sur l entreprise et ouverture sur l autre

# Map (1/2)

- Etape 'Map':

- On doit déterminer la clé à utiliser pour l'opération MAP et écrire son code.
- Puisqu'on s'intéresse aux occurrences des mots dans le texte, et qu'à terme on aura après l'opération REDUCE un résultat pour chacune des clés distinctes, la clé qui s'impose logiquement est : Le mot-lui même.
- Pour l'opération MAP : on va simplement parcourir le fragment qui nous est fourni et, pour chacun des mots, générer le couple clé/valeur: (MOT ; 1). La valeur indique ici l'occurrence pour cette clé, puisqu'on a croisé le mot une fois, on donne la valeur "1".

Le code de l'opération MAP sera donc (ici en pseudo code):

POUR MOT dans LIGNE, FAIRE :  
GENERER COUPLE (MOT; 1)

## Map (2/2)

Les couples (clé;valeur) générés pour chaque fragment sont :

excellence polyvalence et ouverture sont les valeurs de l inpt  
⇒ (excellence;1) (polyvalence;1) (et;1) (ouverture;1) (sont;1) (les;1)  
(valeurs;1) (de;1) (l;1) (inpt;1)

l excellence un challenge  
⇒ (l;1) (excellence;1) (un;1) (challenge;1)

polyvalence et enrichissement des competences  
⇒ (polyvalence;1) (et;1) (enrichissement;1) (des;1) (competences;1)

ouverture sur le monde ⇒ (ouverture;1) (sur;1) (le;1) (monde;1)

ouverture sur l entreprise et ouverture sur l autre  
⇒ (ouverture;1) (sur;1) (l;1) (entreprise;1) (et;1) (ouverture;1) (sur;1)  
(l;1) (autre;1)

# Shuffle (1/2)

- Etape 'Shuffle':
  - Cette opération regroupe tous les couples par clé commune.
  - Elle est effectuée de manière distribuée en utilisant un algorithme de tri distribué, de manière récursive.
  - Après son execution, on obtiendra les 20 groupes suivants :

## Shuffle (2/2)

(excellence;1) (excellence;1)

(et;1) (et;1) (et;1)

(les;1)

(de;1)

(inpt;1)

(challenge;1)

(des;1)

(sur;1) (sur;1) (sur;1)

(monde;1)

(autre;1)

(polyvalence;1) (polyvalence;1)

(sont;1)

(valeurs;1)

(l;1) (l;1) (l;1) (l;1)

(un;1)

(enrichissement;1)

(competences;1)

(le;1)

(entreprise;1)

(ouverture;1) (ouverture;1) (ouverture;1) (ouverture;1)

# REDUCE (1/2)

- Etape 'REDUCE':

- Cette étape sera appelée pour chacun des groupes/clé distincte.
- Dans notre cas, elle va simplement consister à additionner toutes les valeurs liées à la clé spécifiée.
- Le code de l'opération REDUCE sera donc (ici en pseudo code):

```
TOTAL=0
POUR COUPLE dans GROUPE, FAIRE:
    TOTAL=TOTAL+1
    RENVOYER TOTAL
```

## REDUCE (2/2)

Une fois l'opération REDUCE effectuée, on obtiendra donc une valeur unique pour chaque clé distincte. Le résultat sera :

(excellence;2)

(et;3)

(les;1)

(de;1)

(inpt;1)

(challenge;1)

(des;1)

(sur;3)

(monde;1)

(autre;1)

(polyvalence;2)

(sont;1)

(valeurs;1)

(l;4)

(un;1)

(enrichissement;1)

(competences;1)

(le;1)

(entreprise;1)

(ouverture;4)

# Conclusion

- On constate que les mots les plus utilisés dans le texte sont '**I'** et '**'ouverture'** avec 4 occurrences chacun, suivi de 'et' et 'sur' avec 3 occurrences chacun.
- L'exemple choisi est trivial, et son exécution est instantané même sur une machine unique, mais ce qu'il faut savoir c'est qu'il est utile d'utiliser la même implémentation MapReduce sur l'intégralité des textes d'une bibliothèque française, et obtenir ainsi un bon échantillon des mots les plus utilisées dans la langue française.
- L'intérêt du MapReduce est qu'il suffit de développer les opérations Map et Reduce, et de bénéficier automatiquement de la possibilité d'effectuer le traitement sur un nombre variable de machines de manière distribuée.