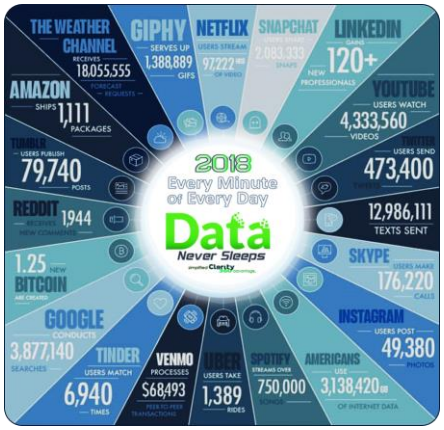
BIG DATA
MAP-REDUCE

CONTEXTE

- Nous traitons de plus en plus de données volumineuses



POURQUOI CE VOLUME DE DONNÉES

- Tendance à tous numériser et sauvegarder pour en tirer des connaissances
- Numérisation des administrations : APC (état civil), Justice (Casier judiciaire), etc.
- Capteurs : température, tension, vent, humidité, etc.
- Smartphones
- Réseaux sociaux, web, etc.
- Données de localisation : GPS, parcours, chemins, etc.
- Données vitales : battements de cœur, respiration, effort, etc.
- Données médicales : rapports, radio, médicaments, etc.
- Objets connectés grâce aux protocoles NFC, Bluetooth, Zigbee : Montres, machines, équipements, etc.
- Etc.

BIG DATA

- Les big data (**mégadonnées**, **données massives**) désigne des **ensembles de données** devenus si **volumineux** qu'ils **dépassent** l'intuition et les **capacités humaines d'analyse** et même celles des **outils informatiques classiques** de gestion de base de données ou de l'informatique,
- L'explosion quantitative (et souvent redondante) de la donnée numérique contraint à de nouvelles manières de voir et analyser le monde,
- De nouveaux problèmes apparaissent

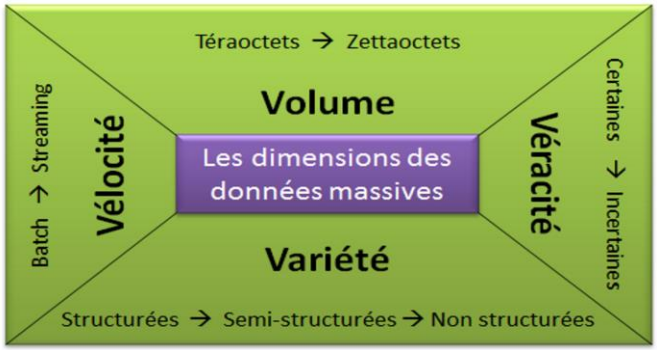
CHALLENGES

- Comment gérer ce volume de données très important
  - Capturer
  - Stocker
  - Rechercher
  - Partager
  - Analyser
  - Visualiser

EMERGENCE DU BIG DATA

- L'utilisation de nouvelles technologies, particulièrement le **Web 2.0**, **les capteurs digitaux**, **les réseaux sociaux**, a donné naissance à des données caractérisées essentiellement par des volumes inhabituels allant au-delà des Téraoctets (***Pétaoctets***, ***Exaoctets*** …)
- Avec tout le buzz autour des données massives, celles-ci sont connues aujourd'hui par le terme **Big Data** et caractérisées par l'expression « **4V** » :
  - ***Volume*** *: taille excessive des données*
  - ***Vélocité*** *: vitesse avec laquelle ces données sont générées/traitées*
  - ***Variété*** *: diversité des formats/structures des données*
  - ***Véracité*** *: problème de fiabilité/précision des données massives*

CARACTERISTIQUES DU BIG DATA: RÉSUMÉ



Les 4V des Big Data

CARACTÉRISTIQUES DU BIG DATA

Volume

- Le volume des données stockées est en pleine expansion
- Les données numériques créées dans le monde : 1,2 zettaoctet/an en 2010, 1,8 en 2011, 2,8 en 2012 et s'élèveront à 40 en 2020.
- En 2013 : Twitter 7 téraoctets de données chaque jour, Facebook 10 téraoctets.
- En 2014: Facebook Hive générait 4 000 To de data par jour

Variété

- Données relationnelles, semi-structurées, non structurées, texte, images, données géo-référencées, etc.

Vélocité

- Fréquence à laquelle les données sont à la fois générées, capturées, partagées et mises à jour
- **Véracité, Valeur :** certaines / incertaines

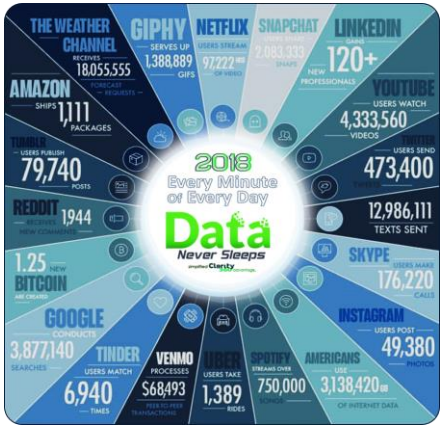
EXEMPLE D'UTILISATION

- Biologie : Séquencement du génome humain
- Prédiction : changements climatiques, dégradation des sols, gaspillage, etc.
- Militaire : mouvement des trafiquants, drones,
- Santé : prédiction, propagation des épidémies
- Politique : analyse d'opinions
- Etc.

INFLUENCE DU BIG DATA

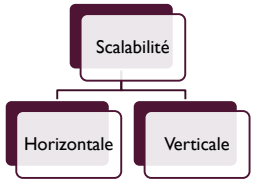
BIG-DATA → HARDWARE

- Nous traitons de plus en plus de données volumineuses
- Les machines actuelles ne suffisent plus pour traiter ce volume de données
- Solution : booster les machines
- ➢ **Scalabilité**



SCALABILITÉ

- **L**a capacité d'un produit **à s'adapter** à un changement **d'ordre de grandeur** de la demande (***montée en charge***), en particulier sa **capacité à maintenir ses fonctionnalités** et ses **performances** en cas de **forte demande,**
- La **scalabilité** est tirée du domaine du génie logiciel pour définir le changement d'échelle dans le cas d'accroissement de charge de données par un système informatique sans pour cela faire impacter ses performances. (solutions hardware et logiciel)



SCALABILITÉ VERTICALE

- **U**tiliser un ordinateur qui offre de nombreuses possibilités **d'ajout de pièces**, sur lequel il est possible de mettre une **grande quantité de mémoire**, de **nombreux processeurs**, plusieurs **cartes mères** et de **nombreux disques durs**.
- Effectuée en effectuant une évolution hardware (CPU plus rapide, plus de RAM, Disques volumineux, etc.
- Limitée par le nombre de CPU, la RAM et les capacités maximum des disques configurées sur une seule machine
- Par exemple un **ordinateur Sun Enterprise** peut contenir jusqu'à **64 processeurs**, **16 cartes mères**, **64 Go de mémoire** et **des baies de stockage**. L'ensemble tout équipé peut coûter jusqu'à **1 million de dollars**

SCALABILITÉ VERTICALE - SUPERCALCULATEURS

- Un **superordinateur** ou **supercalculateur** est un ordinateur conçu pour atteindre les plus hautes performances possibles avec les techniques connues lors de sa conception, en particulier en ce qui concerne la vitesse de calcul.
- La science des superordinateurs : « calcul haute performance » (*High-Performance Computing* ou HPC).
  - Partie *hardware* (conception électronique de l'outil de calcul)
  - Partie *software* (adaptation logicielle du calcul à l'outil).
  - Vitesse en FLOPS (*floating-point operations per second : opération en virgule flottante par seconde),*

Exemples :

1939 : Z2 (Konrad Zuse, Allemagne): 5 FLOPS

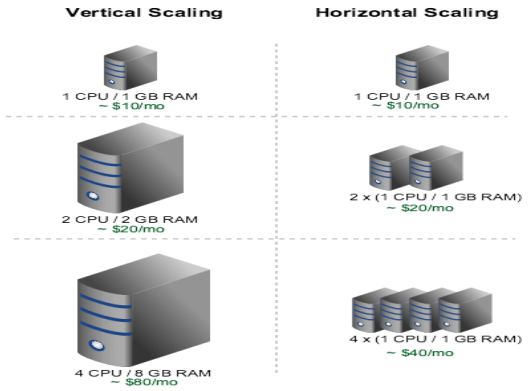
2016 : TaihuLight (NRCPC, Chine), 40 960 Mproc : 93,01 Péta FLOP (10^{15} Flops)

2018 : Behold Summit (IBM/NVIDIA, USA), 36864 : 200 Péta FLOPS,

SCALABILITÉ VERTICALE

- **Avantage**
  - Augmentation des performances avec la même machine
- **Inconvénients**
  - Limitation techniques/technologiques des ressources : RAM, Proc, DD, etc.
  - Augmentation du prix de la solution
  - Accès très limité aux pays développés
- **Solution : Scalabilité Horizontale**

SCALABILITÉ



SCALABILITÉ HORIZONTALE

- Ajouter des ordinateurs « Commodity Hardware » pour faire face à une demande accrue d'un service.
- La méthode la plus courante est la répartition de charge par utilisation d'un Cluster de serveurs.
  - Nécessite une distribution de la BD et la réplication
  - Limitée par les mises à jour et les communications réseaux.

CLUSTERS

- Regrouper plusieurs ordinateurs indépendants appelés nœuds (*node* en anglais), afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur
  - Augmenter la disponibilité ;
  - Faciliter la montée en charge ;
  - Permettre une répartition de la charge ;
  - Faciliter la gestion des ressources (processeur, mémoire vive, disques durs, bande passante réseau).

BIG DATA → PROCESSUS

- Le big data a influencé sur la manière de traitement des données
- Le traitement séquentiel ne suffit plus pour traiter les données volumineuses
- Il faut un nouveau paradigme de programmation qui prend en charge
  - La distribution des données
  - La parallélisation du traitement
  - La réplication des données
  - La montée en charge
- Solution : Map-Reduce

BIG DATA → PROCESSUS

- Le big data a influencé sur la manière de traitement des données
- Le traitement séquentiel ne suffit plus pour traiter les données volumineuses
- Il faut un nouveau paradigme de programmation qui prend en charge
  - La distribution des données
  - La parallélisation du traitement
  - La réplication des données
  - La montée en charge
- **Solution : Map-Reduce**

MAP-REDUCE

- MapReduce est un **modèle de programmation** conçu spécifiquement pour **lire, traiter et écrire** des **volumes de données très importants**.
- Les programmes adoptant ce modèle sont automatiquement **parallélisés** et exécutés sur des **clusters (grappes)** d'ordinateurs.
- MapReduce consiste en deux fonctions **map()** et **reduce()**.

TRAITEMENT DES DONNÉES DISTRIBUÉES

- Google a introduit Map-Reduce
- Ils s'en servent de manière intensive
  - Utilisation en 2007: 400+ TB de données traitées,
  - Temps : 6 min
  - Nombre moyen de machine par traitement Map-Reduce : 436
- L'indexation web traite 20+ TB de données brutes
- Analyse d'images satellites
- Calculs statistiques pour Google translate
- Clustering des Google news
- ....

APPLICATIONS POSSIBLES

- New York Times
  - 4TB d'images TIFF transformées en 11 millions de PDFs
  - Durée 24h, coût 240 $ sur 100 machines EC2
- Caractériser les activités des utilisateurs Facebook
- Analyser les fichiers de logs d'un serveur web
- Fouille de données sur logs des caisses enregistreuses d'un super-marché
- Recherche de séquences d'ADN dans un génôme

MAP-REDUCE

- Un modèle de programmation
- Qui permet
  - Parallélisation automatique
  - Equilibrage de charge
  - Optimisations sur les transferts disques et réseaux
  - Tolérance aux pannes

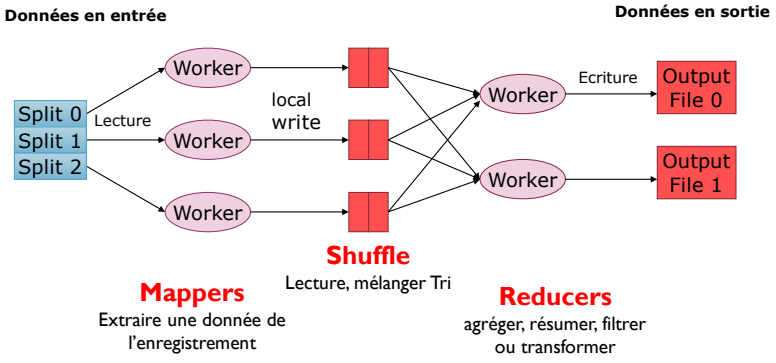
PRINCIPE GÉNÉRAL

1. Lecture des données
2. **Map** : pour chaque élément de données, construire un couple (clé,valeur)
3. **Shuffle :** Trier selon les clés
4. **Reduce**: agréger, résumer, filtrer ou transformer les données
5. Écrire les données

PROBLÈME TYPIQUE TRAITÉ PAR MR

- Lecture d'un volume important de données
- Map: Extraction d'une information utile dans chaque enregistrement
  - Entrée : Texte
  - Sortie : <clé1, val1>, <clé2, val2>, …, <clén, valn>
- Shuffle Mélanger et Trier : {<clé1,val1>}, {<clé2,val2>}, …, {<clén, valn>}
- Reduce: Aggréger, résumer, filtrer ou transformer
  - Entrée : {<cléi,vali>}
  - Sortie : une seule valeur <clé i,Aggr(vali)>
- Ecrire les résultats

MAPREDUCE



27

MAPPERS ET REDUCERS

- Le système créé autant de mappers et de reducer pour garantir un parallélisme important et performant
- Les mappers et reducers s'éxutent entiérement indépendament l'un de l'autre
- Dans Hadoop, les mappers et reducers s'exécutent dans des machines virtuelles différentes,

28

EXEMPLE WORD COUNT

- **Problème** : parmi un ensemble de textes, compter le nombre d'occurrences de chaque mot.
- **Données**: un ensemble de fichiers textes
- **Map** : sur chaque texte, décomposer en mots, et à chaque mot m, ajouter à la liste [(m,1),...]
- **Shuffle**: le système trie/regroupe les paires selon la clé m, dans une liste [(m,[1,1,...]),...]
- **Reduce** : les valeurs 1 sont additionnées pour chaque mot : [(m,2),...]
- **Sortie** : un fichier contenant pour chaque mot(clé) le nombre d'occurences

PSEUDO-CODE

```
Map(Long input_key, String input_values)
Begin
foreach word w in input_values:
EmitIntermediate (w, «1»);
End
Reduce (String key, Iterator intermediate_values)
Begin
int result=0;
foreach v in intermediate_values:
result += ParseInt( v );
Emit (key, String( result ));
End
```

MAPPER - EXEMPLE

- En entrée <Clé,Valeur>
- En sortie ensemble de paires <K',V'>
  - Si on veut compter le nombre de mots dans un post (tweet)
  - L'entrée du mapper sera <PostID, PostText>:

```
<Post1,"The teacher went to the store. The store was closed; the
store opens in the morning. The store opens at 9am." >
```

  - La sortie sera

```
<The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1>
<store,1> <the, 1> <store, 1> <was, 1> <closed, 1>
<the, 1> <store,1> <opens, 1> <in, 1> <the, 1>
<morning, 1> <the 1> <store, 1> <opens, 1> <at, 1>
<9am, 1>
```

31

REDUCER - EXEMPLE

- Reçois les sorties des mappers, il aggrege les valeurs sur la même clé
  - Pour notre exemple, l'entrée su reducer est :

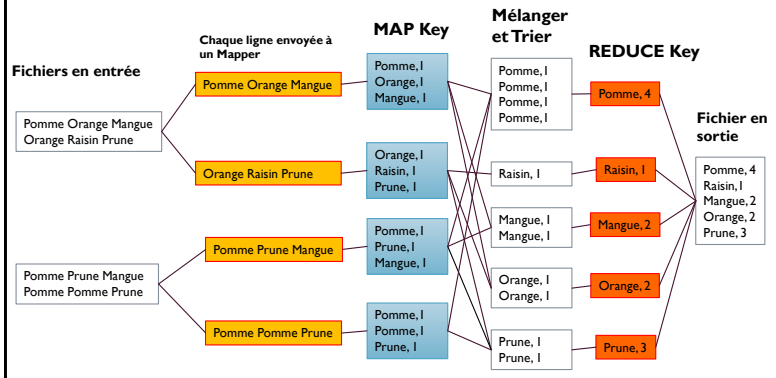
<The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <**store**, 1> <the, 1> <store, 1> <was, 1> <closed, 1> <the, 1> <**store**, 1> <opens,1> <in, 1> <the, 1> <morning, 1> <the 1> <**store**, 1> <opens, 1> <at, 1> <9am, 1>

  - La sortie sera

<The, 6> <teacher, 1> <went, 1> <to, 1> **<store, 3>** <was, 1> <closed, 1> <opens, 1> <morning, 1> <at, 1> <9am, 1>

32

EXAMPLE:WORD COUNT



EXEMPLE 2 INDEX INVERSÉ

- Problème: soit un ensemble de fichiers contenant des mots.
- Etablir une liste des mots, et pour chaque mot une liste des fichiers ou il apparait.

Map(String filename, String line) :

foreach word w in line:

EmitIntermediate(w, filename);

Reduce (String key, Iterator intermediate_values):

// key=word, intermediate_values=filenames

foreach f in intermediate_values:

result += f + ' ';

Emit(key, result));

F1 : Blida Boumerdes Alger
F2 : Tipaza Boumerdes Blida
F3 : Blida Oran Tipaza

Blida : F1 F2 F3
Boumerdes : F1 F2
Alger : F1
Tipaza : F2 F3
Oran : F3

IMPLÉMENTATIONS MAP-REDUCE

- Des librairies MapReduce
- existent pour C++, C#,
- Erlang, Java, Python, Ruby, R
- La plus connue (hors celle propriétaire de Google)
  - Hadoop (projet Apache).

MAP-REDUCE DANS HADOOP

```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<LongWritable, Text, Text, IntWritable>{

    public void map(LongWritable key, Text value, Context context);

          /* Votre code pour map() ici */
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context);

          /* Votre code pour reduce() ici */

    }
  }

  public static void main(String[] args) {
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
  }
}
```

Classe interne héritant de Mapper dont on redéfinit la fonction map

Classe interne héritant de Reducer

Configuration des jobs avant lancement

MAP DANS HADOOP

```
public class Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT>{

    public class Context extends MapContext<KEYIN, VALUEIN, KEYOUT,
    VALUEOUT> {
        // …
    }


    public void map(KEYIN key, VALUEIN value, Context context)
        throws IOException, InterruptedException {
            // …
        }
}
```

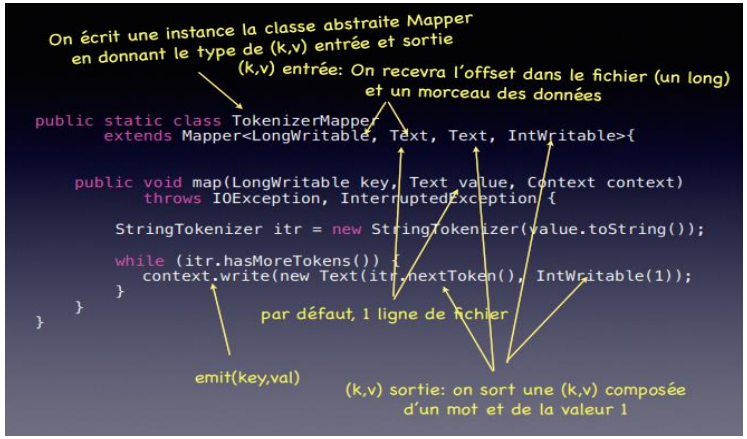

REDUCE DANS HADOOP

```
public class Reducer <KEYIN, VALUEIN, KEYOUT, VALUEOUT> {

    public class Context extends ReducerContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
        // …
    }


    public void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)
        throws IOException, InterruptedException {
            // …
        }
}
```

WORD COUNT SOUS HADOOP


```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<LongWritable, Text, Text, IntWritable>{

    public void map(LongWritable key, Text value, Context context);

          /* Votre code pour map() ici */
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context);

          /* Votre code pour reduce() ici */

    }
  }

  public static void main(String[] args) {
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
  }
}
```


FONCTION MAP




```
public static class TokenizerMapper
       extends Mapper<LongWritable, Text, Text, IntWritable>{

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

       StringTokenizer itr = new StringTokenizer(value.toString());

       while (itr.hasMoreTokens()) {
          context.write(new Text(itr.nextToken()), IntWritable(1));
       }
    }
}
```

FONCTION REDUCE

```
public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {

   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
             sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```


INPUTSPLIT ET MAP

- Lorsqu'un Job MapReduce est démarré pour traiter un fichier stocké dans HDFS , l'une des choses qu'Hadoop fait est de diviser l'entrée en divisions logiques, ces divisions sont appelées **input splits dans Hadoop** .
- Un **InputSplit (ou split)** représente un morceau des données traitée par une opération map.
- Le nombre de mappeurs démarrés est égal au nombre de InputSplit,
- Chaque map traite un seul split.
- Par exemple, si les données d'entrée sont divisées logiquement en 8 inputsplit, 8 mappeurs seront démarrés pour traiter ces inputsplit en parallèle.
- Chaque inputsplit est divisé en records de type (key,value) et le mappeur traite successivement chaque record.

INPUTFORMAT

- **Dans Hadoop**
  - La classe **InputFormat** divise les fichiers d'entrée en InputSplits logiques.
  - La **classe RecordReader** divise les données en paires clé / valeur qui sont ensuite transmises en entrée au mappeur.
- **FileInputFormat** est la classe de base pour toutes les implémentations de InputFormat.
- **FileInputFormat** fournit essentiellement les méthodes **addInputPath()** et **addInputPaths()** pour spécifier des chemins de fichiers à envoyer en entrée d'un job map-reduce
- **Exemple**
  - **L**a classe **TextInputFormat**, lit un fichier texte en le décomposant en un ensemble de lignes
  - **KeyValueInputFormat** : transforme les données en entrée en un ensemble de <clé, valeur>

EXEMPLE OUTPUT FORMAT

- **OutputFormat** décrit la spécification de sortie pour un travail Map-Reduce.
- Map-Reduce repose sur OutputFormat pour:
  - Validez la spécification de sortie du job. Par exemple, vérifiez que le répertoire de sortie n'existe pas déjà.
  - Utiliserla classe **RecordWriter** pour écrire les fichiers de sortie <Clé,Valeur> du job. Les fichiers de sortie sont stockés dans un fichier HDFS.

COMPILER ET EXECUTER (HADOOP)

- **Set $JAVA_HOME and $HADOOP_CLASS**
  - *export HADOOP_VERSION=2.4.0*
  - *export HADOOP_HOME=/usr/local/Cellar/hadoop/${HADOOP_VERSION}*
  - *export PATH=$HADOOP_HOME/bin:$PATH*
  - *export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar:.*
- **Compiler le projet**
  - *hadoop com.sun.tools.javac.Main WordCount.java*
- **Construire le fichier .JAR**
  - jar -cvf wc.jar *.class
- **Exécuter**
  - *hadoop jar wc.jar WordCount input output*