FUN FRANCE UNIVERSITÉ NUMÉRIQUE

Université Nice Sophia Antipolis

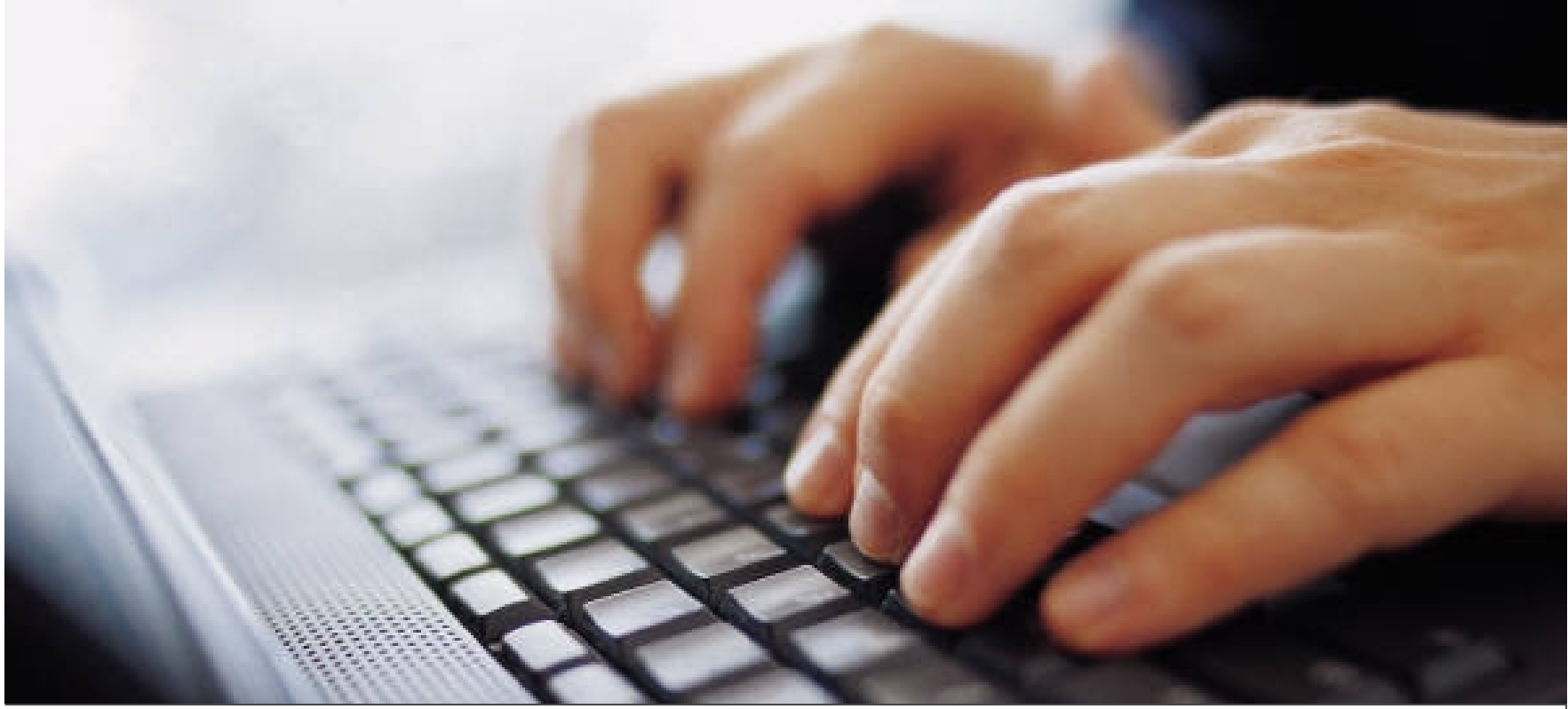
Introduction à Hadoop & MapReduce

Cours 2

Benjamin Renaut <renaut.benjamin@tokidev.fr>

MOOC / FUN

2014 - 2015



5 Hadoop: présentation

Apache Hadoop

- **Projet Open Source – fondation Apache.**
  http://hadoop.apache.org/

- **Développé en Java; portable.**

- **Assure l'exécution de tâches map/reduce; fourni le *framework*, en Java, pour leur développement.**

- **Autres langages supportés par le biais d'utilitaires internes.**

Historique

- 2003/2004: publication par Google de deux *whitepapers*, le premier sur GFS (un système de fichier distribué) et le second sur le paradigme Map/Reduce pour le calcul distribué.

- 2004: développement de la première version du *framework* qui deviendra Hadoop par Doug Cutting (archive.org).

- 2006: Doug Cutting (désormais chez Yahoo) développe une première version exploitable de Apache Hadoop pour l'amélioration de l'indexation du moteur de recherche. Encore primitif (au maximum quelques machines, etc.).

Historique

- 2008: développement maintenant très abouti, Hadoop utilisé chez Yahoo dans plusieurs départements.

- 2011: Hadoop désormais utilisé par de nombreuses autres entreprises et des universités, et le *cluster* Yahoo comporte 42000 machines et des centaines de peta-octets d'espace de stockage.

Le nom lui-même n'est pas un acronyme: il s'agit du nom d'un éléphant en peluche du fils de l'auteur originel (Doug Cuttin) de Hadoop.

Qui utilise Hadoop

… et des milliers d'entreprises et universités à travers le monde.

Le Big Data

- **La multiplication des données, dans un volume toujours plus important, et leur traitement, les problématiques posées et toutes les nouvelles possibilités et usages qui en découlent sont couverts par l'expression « Big Data ».**

- **Hadoop est au coeur de ces problématiques; c'est (de très loin) le logiciel/*framework* le plus utilisé pour y répondre.**

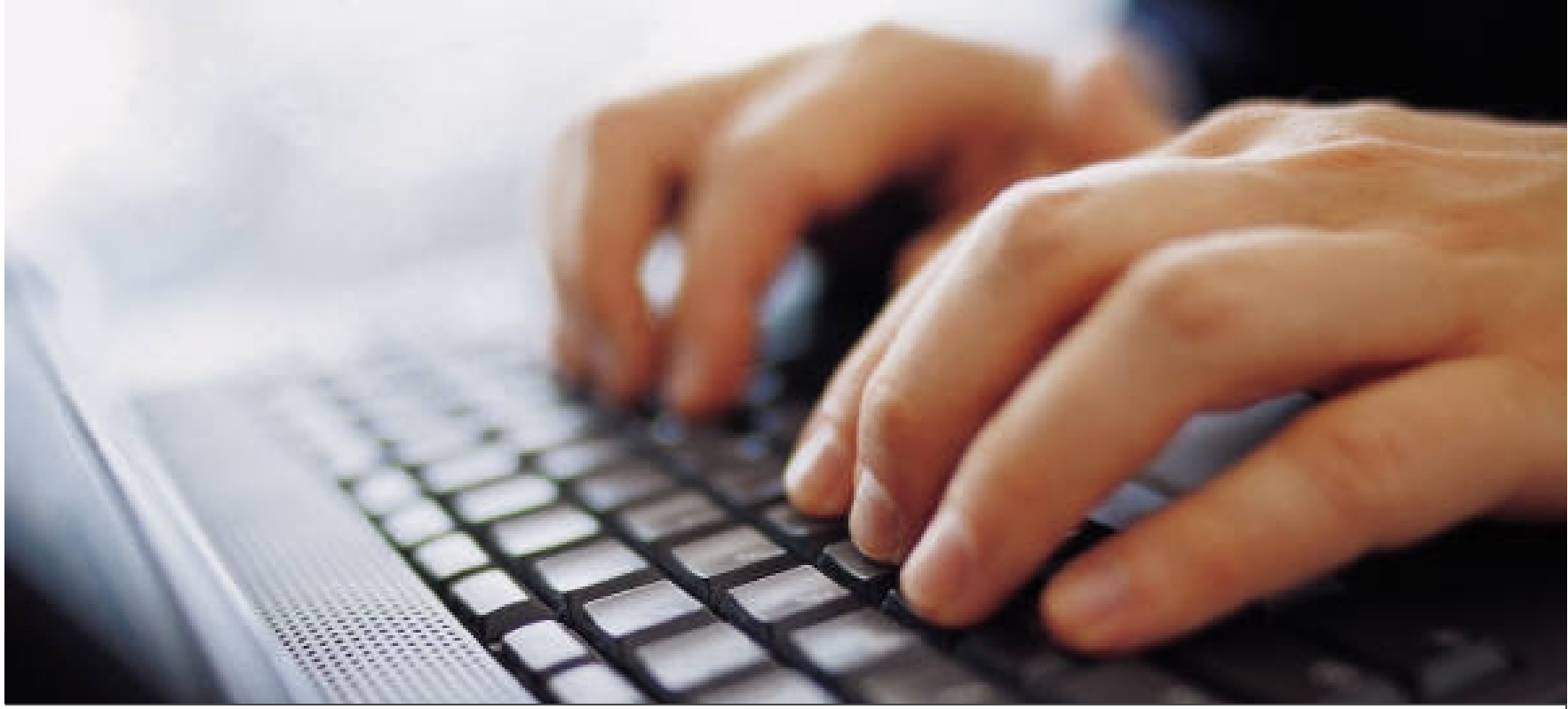
Avantages et inconvénients

- **Hadoop est:**
  - **Simple à utiliser et à déployer.**
  - **Portable (clusters hétérogènes).**
  - **Indéfiniment *scalable*.**
  - **Très extensible et interconnectable avec d'autres solutions.**
- **Il a également des inconvénients: *overhead* supérieur à une solution propriétaire spécifique dédiée à un problème; encore expérimental et en développement...**

Conclusion

- **Hadoop n'utilise aucun principe foncièrement nouveau; il offre en revanche une très forte simplicité et souplesse de déploiement inconnues jusqu'à présent pour l'exécution facile de tâches parallèles variées.**

- **Grâce à Hadoop, même des structures limitées en taille/ressources peuvent facilement avoir accès à de fortes capacités de calcul: déploiement à bas coût de *clusters* en interne ou location de temps d'exécution *via* les services de *cloud computing*.**



1 HDFS

Présentation

- **HDFS: Hadoop Distributed File System.**
- **Système de fichiers distribué associé à Hadoop. C'est là qu'on stocke données d'entrée, de sortie, etc.**
- **Caractéristiques:**
  - **Distribué**
  - **Redondé**
  - **Conscient des caractéristiques physiques de l'emplacement des serveurs (*racks*) pour l'optimisation.**

Architecture

- **Repose sur deux serveurs:**
  - **Le *NameNode*, unique sur le *cluster*. Stocke les informations relative aux noms de fichiers et à leurs caractéristiques de manière centralisée.**
  - **Le *DataNode*, plusieurs par *cluster*. Stocke le contenu des fichiers eux-même, fragmentés en blocs (64KB par défaut).**
- **Inspiré de GFS, lui-même issu de recherches de Google. « *The Google File System* », 2003.**

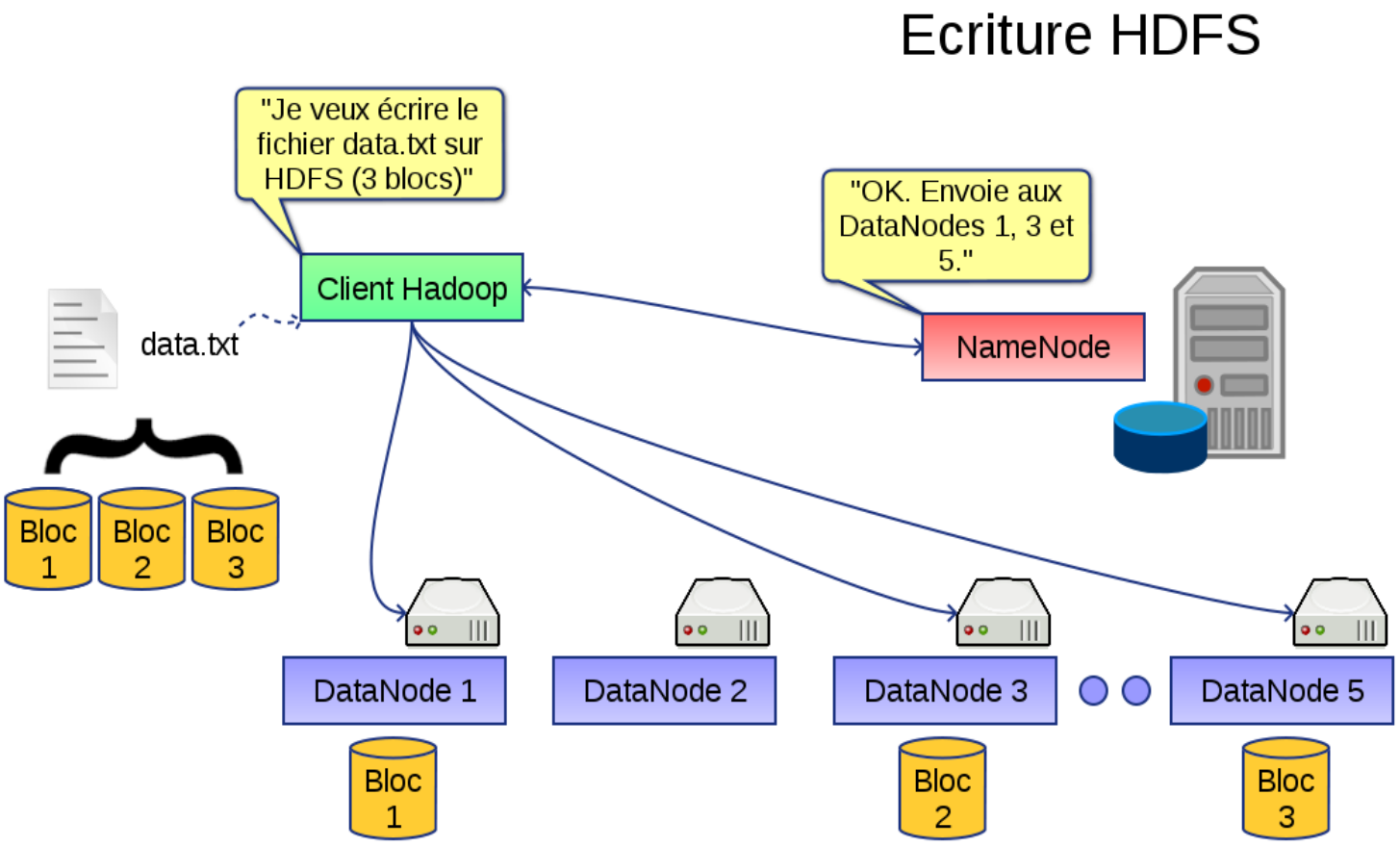
Architecture

- **Repose sur deux serveurs:**
  - **Le *NameNode*, unique sur le *cluster*. Stocke les informations relative aux noms de fichiers et à leurs caractéristiques de manière centralisée.**
  - **Le *DataNode*, plusieurs par *cluster*. Stocke le contenu des fichiers eux-même, fragmentés en blocs (64KB par défaut).**
- **Inspiré de GFS, lui-même issu de recherches de Google. « *The Google File System* », 2003.**

Écriture d'un fichier

- **Pour écrire un fichier:**
  - **Le client contacte le *NameNode* du cluster, indiquant la taille du fichier et son nom.**
  - **Le *NameNode* confirme la demande et indique au client de fragmenter le fichier en blocs, et d'envoyer tel ou tel bloc à tel ou tel DataNode.**
  - **Le client envoie les fragments aux DataNode.**
  - **Les DataNodes assurent ensuite la réplication des blocs.**

Écriture d'un fichier

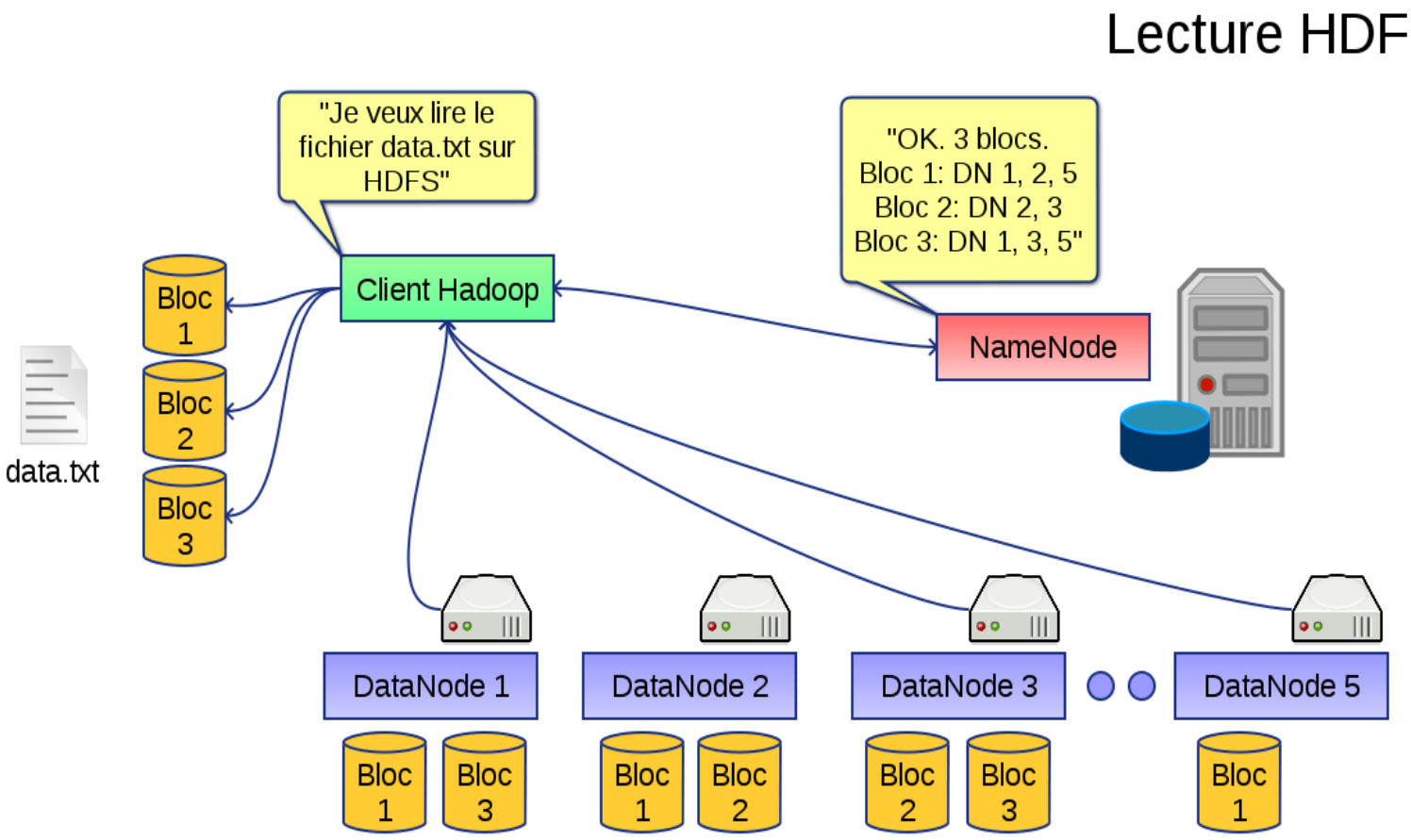


- Le client indique au NameNode qu'il souhaite écrire un bloc.
- Celui-ci lui indique le DataNode à contacter.
- Le client envoie le bloc au Datanode.
- Les DataNodes répliquent le bloc entre eux.
- Le cycle se répète pour le bloc suivant.

Lecture d'un fichier

- **Pour lire un fichier:**
  - **Le client contacte le *NameNode* du cluster, indiquant le fichier qu'il souhaite obtenir.**
  - **Le *NameNode* lui indique la taille, en blocs, du fichier, et pour chaque bloc une liste de DataNodes susceptibles de lui fournir.**
  - **Le client contacte les DataNodes en question pour obtenir les blocs, qu'il reconstitue sous la forme du fichier.**
  - **En cas de DataNode inaccessible/autre erreur pour un bloc, le client contacte un DataNode alternatif de la liste pour l'obtenir.**

Lecture d'un fichier



- Le client indique au NameNode qu'il souhaite lire un fichier.
- Celui-ci lui indique sa taille et les différents DataNode contenant les N blocs.
- Le client récupère chacun des blocs à un des DataNodes.
- Si un DataNode est indisponible, le client le demande à un autre.

En pratique

- **On utilise la commande console `hadoop`, depuis n'importe quelle machine du *cluster*.**

- **Sa syntaxe reprend celle des commandes Unix habituelles:**

```
$ hadoop fs -put data.txt /input/files/data.txt
$ hadoop fs -get /input/files/data.txt data.txt
$ hadoop fs -mkdir /output
$ hadoop fs -rm /input/files/data.txt
… etc …
```

Remarques

- **L'ensemble du système de fichier virtuel apparaît comme un disque « unique »: on ne se soucie pas de l'emplacement réel des données.**
- **Tolérant aux pannes: blocs répliqués.**
- **Plusieurs *drivers* FUSE existent pour monter le système de fichier HDFS d'une manière plus « directe ».**

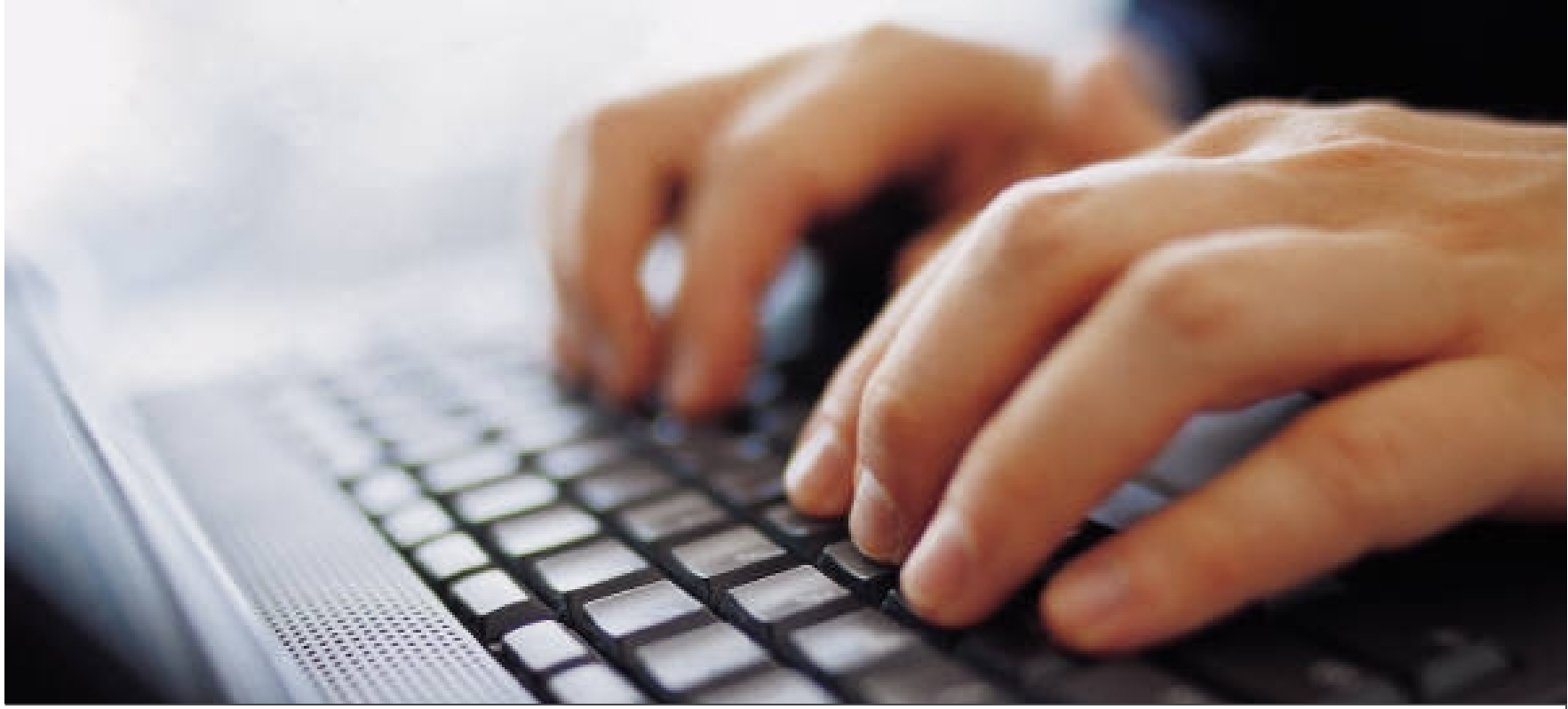
Inconvénients

- **NameNode unique: si problème sur le serveur en question, HDFS est indisponible. Compensé par des architectures serveur active/standby.**

- **Optimisé pour des lectures concurrentes; sensiblement moins performant pour des écritures concurrentes.**

Alternatives

- **D'autres systèmes de fichiers distribués (Amazon S3, Cloudstore…).**
- **Ponts d'interconnexion directe avec des systèmes de bases de données relationnelles (Oracle, MySQL, PostGreSQL).**
- **Autres ponts d'interconnexion (HTTP, FTP).**



2 Hadoop: architecture

Architecture

- **Repose sur deux serveurs:**
  - **Le *JobTracker*, unique sur le *cluster*. Reçoit les tâches map/reduce à exécuter (sous la forme d'une archive Java .jar), organise leur exécution sur le *cluster*.**
  - **Le *TaskTracker*, plusieurs par *cluster*. Exécute le travail map/reduce lui-même (sous la forme de tâches map et reduce ponctuelles avec les données d'entrée associées).**
- **Chacun des TaskTrackers constitue une unité de calcul du *cluster*.**

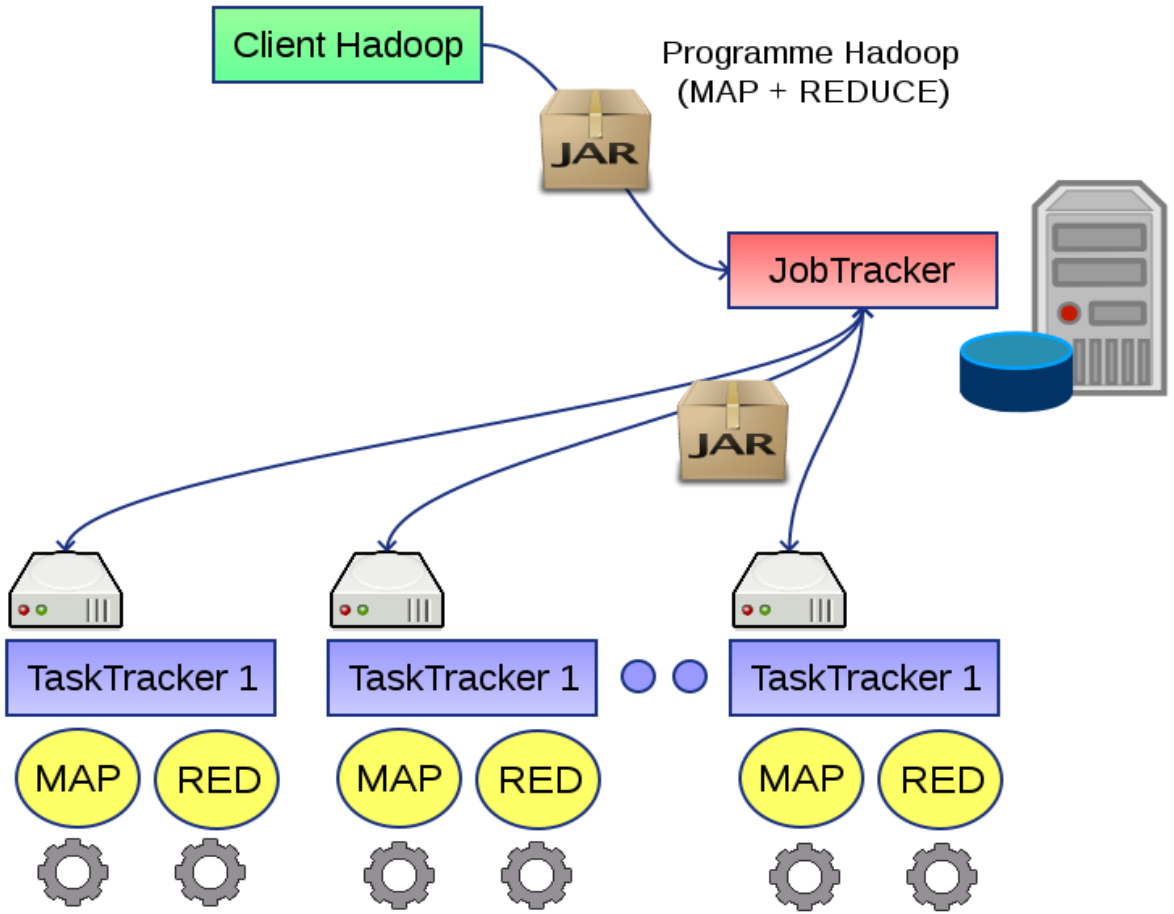
Architecture

- **Le serveur *JobTracker* est en communication avec HDFS; il sait où sont les données d'entrée du programme map/reduce et où doivent être stockées les données de sortie. Il peut ainsi optimiser la distribution des tâches selon les données associées.**

- **Pour exécuter un programme map/reduce, on devra donc:**

  - **Écrire les données d'entrée sur HDFS.**
  - **Soumettre le programme au JobTracker du cluster.**
  - **Récupérer les données de sortie depuis HDFS.**

Architecture



Exécution d'une tâche

- **Tous les TaskTrackers signalent leur statut continuellement par le biais de paquets *heartbeat*.**

- **En cas de défaillance d'un TaskTracker (heartbeat manquant ou tâche échouée), le JobTracker avise en conséquence: redistribution de la tâche à un autre nœud, etc.**

- **Au cours de l'exécution d'une tâche, on peut obtenir des statistiques détaillées sur son évolution (étape actuelle, avancement, temps estimé avant completion, etc.), toujours par le biais du client console `hadoop`.**

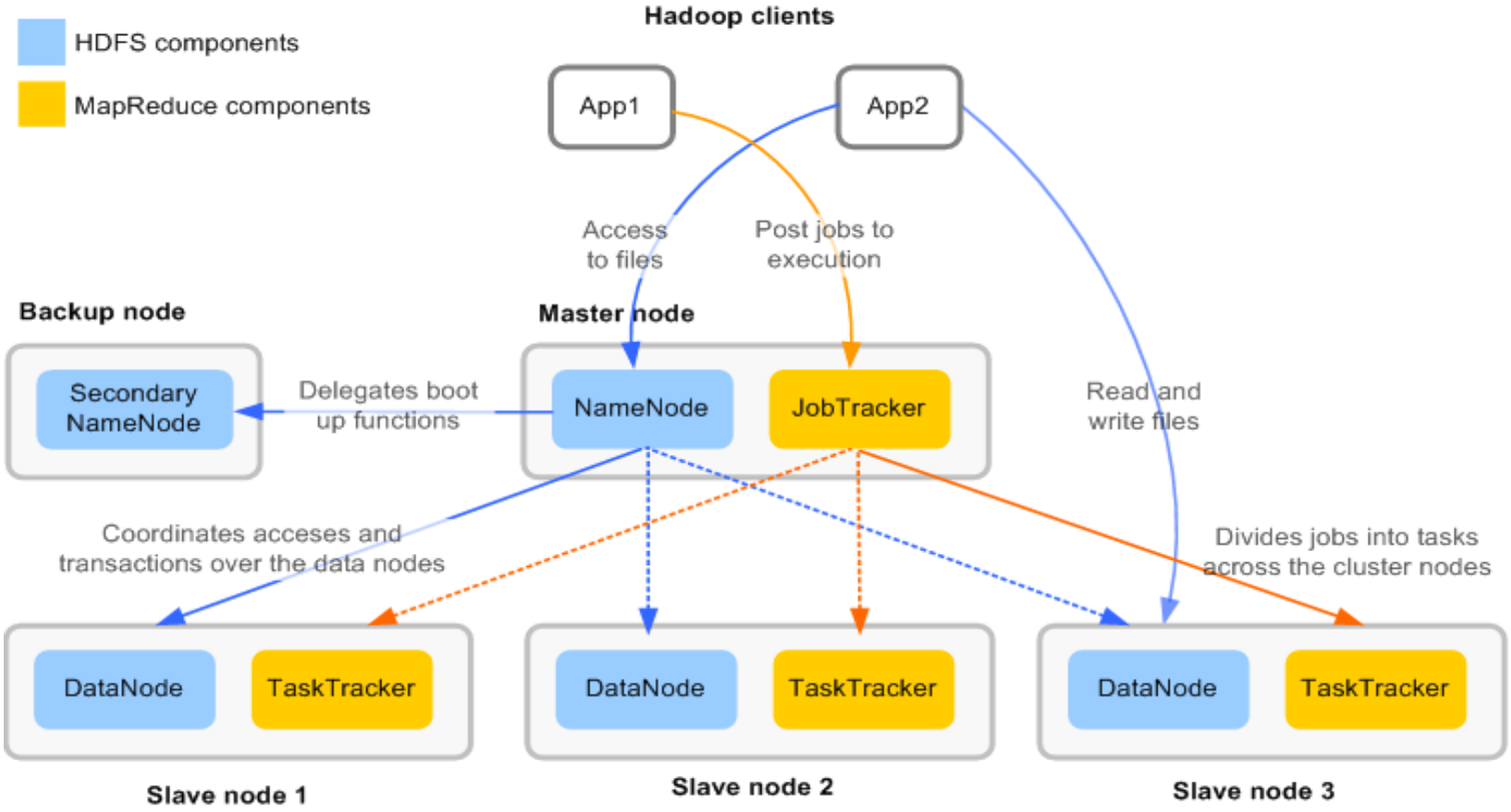
Autres considérations

- **Un seul JobTracker sur le serveur: point de défaillance unique. Là aussi, compensé par des architectures serveurs adaptées.**

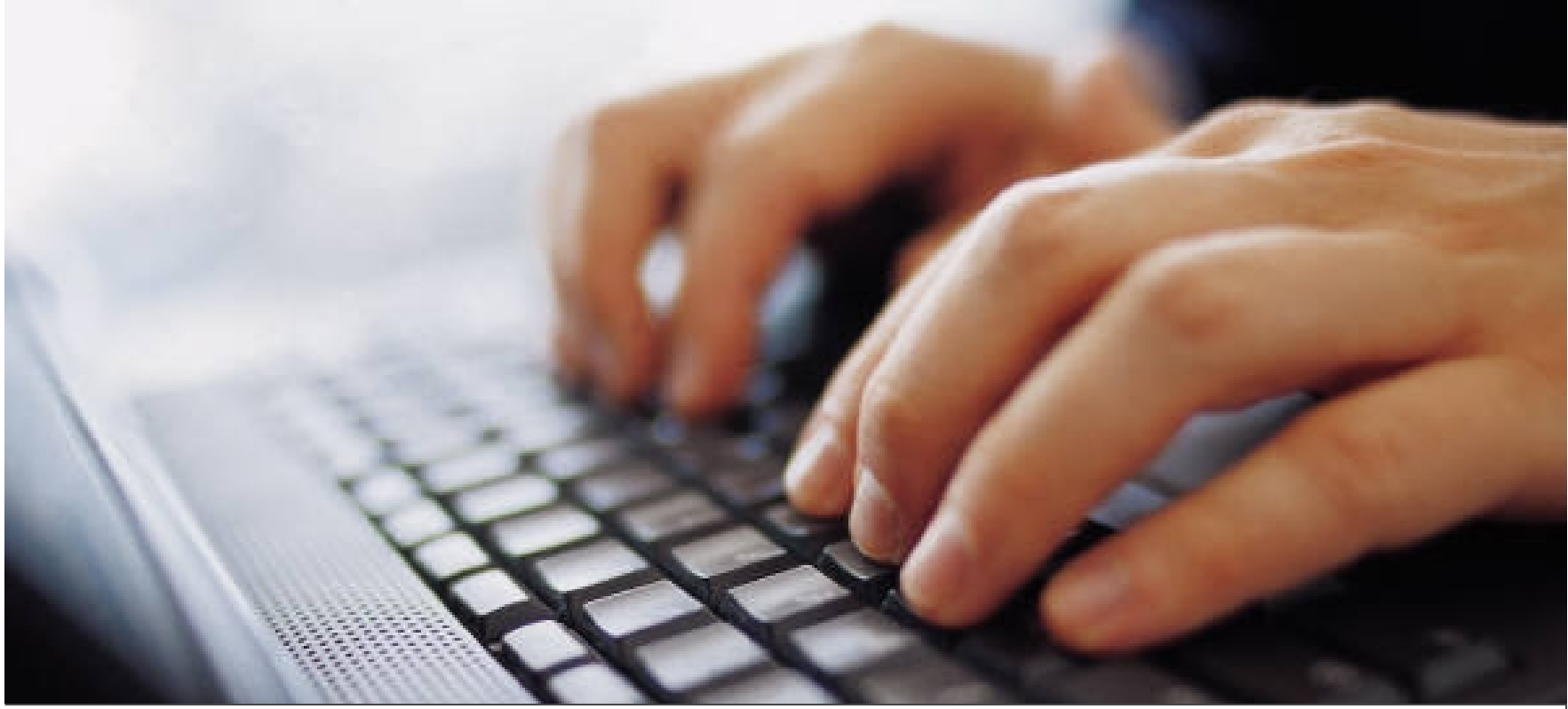
- **Les deux serveurs « uniques » NameNode et JobTracker sont souvent actifs au sein d'une seule et même machine: le nœud maître du *cluster*.**

- **Tout changement dans la configuration du *cluster* est répliqué depuis le nœud maître sur l'intégralité du *cluster*.**

Architecture générale



Source: documentation Hadoop



3 Hadoop: installation

Objectif

- **Couvrir l'installation de Hadoop dans son environnement typique de déploiement (serveurs GNU/Linux ou similaires).**

- **Couvrir la configuration d'une instance Hadoop vous permettant d'expérimenter avec son utilisation et d'une manière plus générale avec map/reduce.**

- **Hadoop est capricieux sur environnement Windows; il est encore très expérimental => utiliser une machine virtuelle si nécessaire pour l'expérimentation.**

Installation de Hadoop

- **Deux options:**
  - **Par le biais de paquets adaptés à la distribution (.deb pour Debian/Ubuntu/etc., .rpm pour Red Hat/Centos/etc.).**
  - **Par le biais d'un *tarball* officiel de la fondation Apache: installation « manuelle ».**

Installation *via* paquet

- **Si disponible, utiliser les outils standards de la distribution: apt-get / yum.**

- **Alternativement, Cloudera, une entreprise phare de solutions Big Data / support et développement Hadoop, met à disposition sa propre distribution, CDH:**

  http://www.cloudera.com/content/cloudera/en/downloads/cdh.html

Installation manuelle

- **Pré-requis:**
  - **Distribution GNU/Linux moderne.**
  - **Java (version Sun ou OpenJDK).**
- **On commence par créer le groupe et l'utilisateur qui seront spécifiques à Hadoop:**

```
# addgroup hadoop
# adduser --ingroup hadoop hadoopuser
# adduser hadoopuser
```

Installation manuelle

- **On télécharge ensuite Hadoop, en l'installant dans un répertoire au sein de /opt:**

```
# wget http://MIROIR/hadoop/hadoop-X.Y.Z.tar.gz
# tar vxzf hadoop-X.Y.Z.tar.gz -C /opt
# cd /opt
# ln -s /opt/hadoop-X.Y.Z /opt/hadoop
# chown -R hadoopuser:hadoop /opt/hadoop*
```

(remplacer l'URL par un des miroirs et ajuster le numéro de version)

Installation manuelle

- **Il faut ensuite ajouter les différentes déclarations suivantes au sein du fichier .bashrc de l'utilisateur hadoopuser:**

```
# Variables Hadoop. Modifier l'emplacement du SDK java.
export JAVA_HOME=/path/to/java/jdk
export HADOOP_INSTALL=/opt/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_HOME=$HADOOP_INSTALL
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export HADOOP_YARN_HOME=$HADOOP_INSTALL
export HADOOP_CONF_DIR=$HADOOP_INSTALL/etc/hadoop
```

Installation manuelle

- **Enfin, il est aussi nécessaire d'indiquer l'emplacement du JDK Java dans le fichier /opt/hadoop/etc/hadoop/hadoop-env.sh. Localiser la ligne « export JAVA_HOME » au sein de ce fichier et l'éditer de telle sorte qu'elle pointe sur l'emplacement du JDK Java:**

```
export JAVA_HOME=/path/to/java/jdk
```

- **Sauvegarder. Hadoop est désormais installé. Pour le confirmer, on peut par exemple utiliser la commande « hadoop version ».**

Installation manuelle – Configuration

- **Il faut maintenant configurer Hadoop en mode nœud unique (pour l'expérimentation).**

- **Editer le fichier /opt/hadoop/etc/hadoop/core-site.xml, et insérer entre les deux balises <configuration> les lignes suivantes:**

```
<property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:9000</value>
</property>
```

Installation manuelle – Configuration

- **Editer maintenant le fichier /opt/hadoop/etc/hadoop/yarn-site.xml, et insérer entre les deux balises <configuration> les lignes suivantes:**

```
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>
<property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Installation manuelle – Configuration

- **Renommer maintenant le fichier mapred-site.xml.template, au sein du même répertoire /opt/hadoop/etc/hadoop:**

```
$ cd /opt/hadoop/etc/hadoop
$ mv mapred-site.xml.template mapred-site.xml
```

- **Et ajouter les lignes suivantes entre les deux balises <configuration> au sein du fichier renommé mapred-site.xml:**

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
```

Installation manuelle – Configuration

- **Il faut maintenant créer les répertoires où seront stockés les données HDFS:**

```
# mkdir -p /opt/hdfs/namenode
# mkdir -p /opt/hdfs/datanode
# chown -R hadoopuser:hadoop /opt/hdfs
```

- **Editer ensuite le fichier:**

  /opt/hadoop/etc/hadoop/hdfs-site.xml

Installation manuelle – Configuration

- **… et insérer les lignes suivantes entre les deux balises <configuration>:**

```
<property>
   <name>dfs.replication</name>
   <value>1</value>
</property>
<property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/opt/hdfs/namenode</value>
</property>
<property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/opt/hdfs/datanode</value>
</property>
```

Installation manuelle – Configuration

- **Il reste enfin à formater le système de fichiers HDFS local:**

```
$ hdfs namenode -format
```

- **Hadoop est désormais correctement installé et configuré (un seul nœud). Le démarrer avec les commandes:**

```
$ start-dfs.sh
$ start-yarn.sh
```

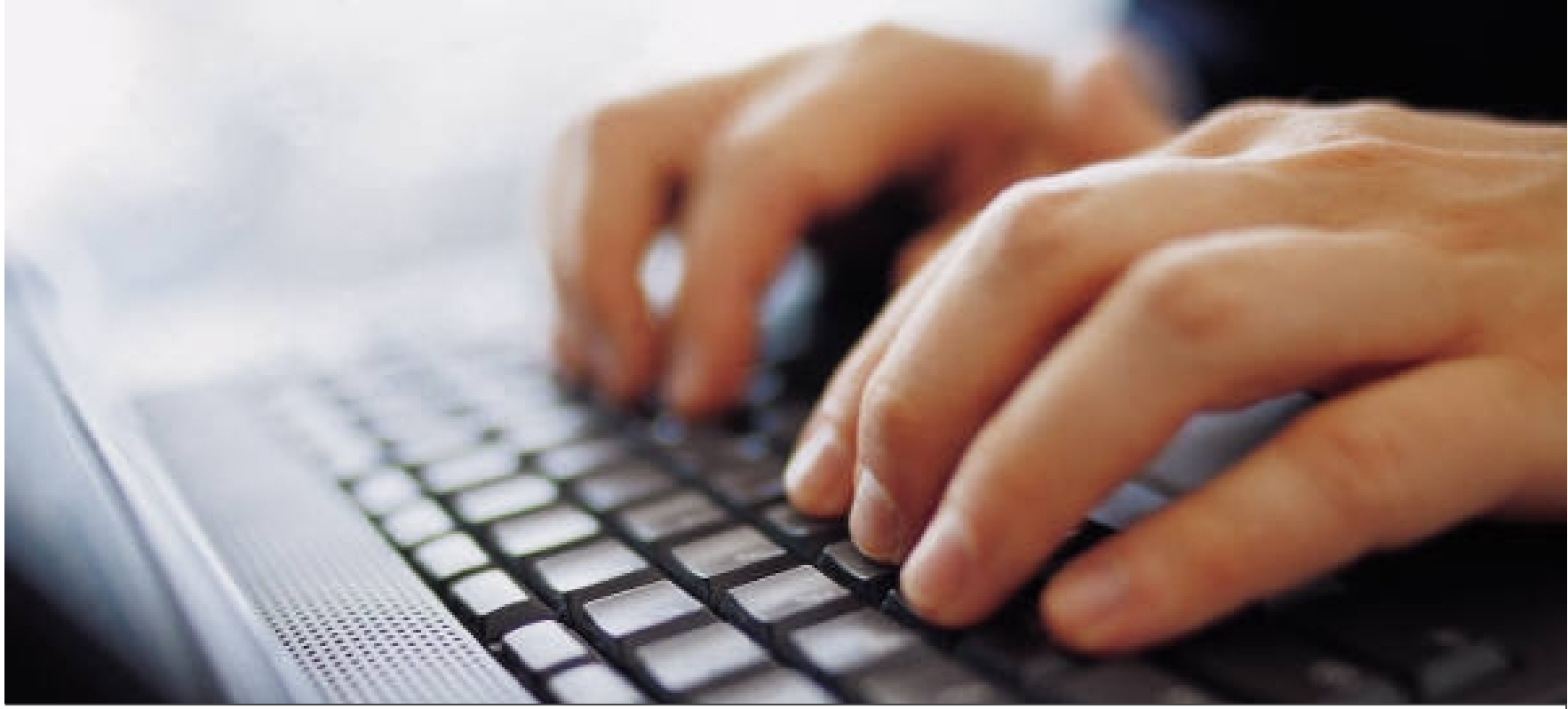
- **Les commandes et codes présentés dans les sections successives du cours peuvent être testés sur une telle installation.**

Machine virtuelle

- **Au besoin, une machine virtuelle associée au présent cours est par ailleurs disponible à l'adresse:**

  http://mooc.tokidev.fr/hadoop_vm_2014.ova

  (importable au sein de VirtualBox / VMWare).



4 Hadoop: utilisation (1)

API Java

- **Un programme Hadoop « natif » est développé en Java, en utilisant le SDK Hadoop, basé sur le mécanisme d'interfaces.**

- **Trois classes au minimum au sein d'un programme Hadoop:**

  - **Classe Driver: *Main* du programme. Informe Hadoop des différents types et classes utilisés, des fichiers d'entrée/de sortie, etc.**
  - **Classe Mapper: implémentation de la fonction MAP.**
  - **Classe Reducer: implémentation de la fonction REDUCE.**

API Java – Classe Driver

- **La classe Driver doit au minimum:**
  - **Passer à Hadoop des arguments « génériques » de la ligne de commande, par le biais d'un objet Configuration.**
  - **Informer Hadoop des classes Driver, Mapper et Reducer par le biais d'un objet Job; et des types de clef et de valeur utilisés au sein du programme map/reduce par le biais du même objet.**
  - **Spécifier à Hadoop l'emplacement des fichiers d'entrée sur HDFS; spécifier également l'emplacement où stocker les fichiers de sortie.**
  - **Lancer l'exécution de la tâche map/reduce; recevoir son résultat.**

En pratique

- **Création de l'objet Configuration:**

```
Configuration conf=new Configuration();
```

- **Passage à Hadoop de ses arguments « génériques »:**

```
String[] ourArgs=new GenericOptionsParser(conf, args).getRemainingArgs();
```

- **On créé ensuite un nouvel objet Job:**

```
Job job=Job.getInstance(conf, "Compteur de mots v1.0");
```

- **On indique à Hadoop les classes Driver, Mapper et Reducer:**

```
job.setJarByClass(WCount.class);
job.setMapperClass(WCountMap.class);
job.setReducerClass(WCountReduce.class);
```

En pratique

- **On indique ensuite quels sont les types de clef/valeur utilisés au sein du programme:**

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

il s'agit ici de types Hadoop.

- **On spécifie l'emplacement des fichiers d'entrée/de sortie:**

```
FileInputFormat.addInputPath(job, new Path("/data/input.txt"));
FileOutputFormat.setOutputPath(job, new Path("/data/results));
```

(ici *via* HDFS)

En pratique

- **Enfin, il reste à exécuter la tâche elle-même:**

```
if(job.waitForCompletion(true))
  System.exit(0);
System.exit(-1);
```

- ***Packages* Java à importer depuis l'API Hadoop, dans cet exemple:**

```
org.apache.hadoop.mapreduce.Job
org.apache.hadoop.mapreduce.lib.input.FileInputFormat
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat
org.apache.hadoop.conf.Configuration
org.apache.hadoop.util.GenericOptionsParser
org.apache.hadoop.fs.Path
org.apache.hadoop.io.Text
org.apache.hadoop.io.IntWritable
```

Remarques

- **Il existe d'autres classes permettant de spécifier une source pour les données d'entrée/une destination pour les données de sortie; c'est la manière dont fonctionnent les alternatives à HDFS.**

- **On peut spécifier des types (clef;valeur) différents pour la sortie de l'opération MAP et la sortie de l'opération REDUCE (job.setMapOutputKeyClass / job.setMapOutputValueClass).**

- **Il s'agit là du strict minimum pour une classe Driver; on aura généralement des implémentations plus complexes.**

La classe Mapper

- **La classe Mapper est en charge de l'implémentation de la méthode map du programme map/reduce.**

- **Elle doit étendre la classe Hadoop org.apache.hadoop.mapreduce.Mapper. Il s'agit d'une classe générique qui se paramétrise avec quatre types:**

  - **Un type *keyin*: le type de clef d'entrée.**
  - **Un type *valuein*: le type de valeur d'entrée.**
  - **Un type *keyout*: le type de clef de sortie.**
  - **Un type *valueout*: le type de valeur de sortie.**

La classe Mapper

- On déclarera une classe Mapper par exemple ainsi (ici pour l'exemple du compteur d'occurences de mots):

```
public class WCountMap extends Mapper<Object, Text, Text, IntWritable>
```

- C'est la méthode *map* qu'on doit implémenter. Son prototype:

```
protected void map(Object key, Text value, Context context)
  throws IOException, InterruptedException
```

Elle est appelée pour chaque couple (clef;valeur) d'entrée: respectivement les arguments key et value.

La classe Mapper

- **Le troisième argument, *context*, permet entre autres de renvoyer un couple (clef;valeur) en sortie de la méthode map. Par exemple:**

```
context.write("ciel", 1);
```

… il a d'autres usages possibles (passer un message à la classe Driver, etc.).

Il faut évidemment que la clef et la valeur renvoyées ainsi correspondent aux types keyout et valueout de la classe Mapper.

La classe Mapper

- **Les différents *packages* Java à importer depuis l'API Hadoop:**

```
org.apache.hadoop.mapreduce.Job
org.apache.hadoop.io.Text
org.apache.hadoop.io.IntWritable
org.apache.hadoop.mapreduce.Mapper
```

La classe Reducer

- **Similaire à la classe Mapper, elle implémente la méthode reduce du programme map/reduce.**

- **Elle doit étendre la classe Hadoop org.apache.hadoop.mapreduce.Reducer. Il s'agit là aussi d'une classe générique qui se paramétrise avec les mêmes quatre types que pour la classe Mapper: keyin, valuein, keyout et valueout.**

- **A noter que contrairement à la classe Mapper, la classe Reducer recevra ses arguments sous la forme d'une clef unique et d'une liste de valeurs correspondant à cette clef.**

La classe Reducer

- **On déclarera une classe Reducer par exemple ainsi:**

```
public class WCountReduce extends Reducer<Text, IntWritable, Text, IntWritable>
```

- **C'est la méthode *reduce* qu'on doit implémenter. Son prototype:**

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

La fonction est appelée une fois par clef distincte.
C'est l'argument *key* qui contient la clef distincte, et l'argument Iterable Java *values* qui contient la liste des valeurs correspondant à cette clef.

La classe Reducer

- **Là aussi, on peut renvoyer un couple (clef;valeur) en sortie de l'opération reduce par le biais de la méthode *write* de l'argument *context*:**

```
context.write("ciel", 5);
```

- **Les différents *packages* Java à importer:**

```
org.apache.hadoop.mapreduce.Job
org.apache.hadoop.io.Text
org.apache.hadoop.io.IntWritable
import org.apache.hadoop.mapreduce.Reducer;
```

Exemple – Occurences de mots


```
// Le main du programme.
public static void main(String[] args) throws Exception
{
    // Créé un object de configuration Hadoop.
    Configuration conf=new Configuration();

    // Permet à Hadoop de lire ses arguments génériques,
     // récupère les arguments restants dans ourArgs.
    String[] ourArgs=new GenericOptionsParser(conf,
                          args).getRemainingArgs();

    // Obtient un nouvel objet Job: une tâche Hadoop. On
   // fourni la configuration Hadoop ainsi qu'une description
    // textuelle de la tâche.
    Job job=Job.getInstance(conf, "Compteur de mots v1.0");
```

Exemple – Occurences de mots


```
// Défini les classes driver, map et reduce.
job.setJarByClass(WCount.class);
job.setMapperClass(WCountMap.class);
job.setReducerClass(WCountReduce.class);

// Défini types clefs/valeurs de notre programme Hadoop.
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

// Défini les fichiers d'entrée du programme et le
 // répertoire des résultats. On se sert du premier et du
 // deuxième argument restants pour permettre à
 // l'utilisateur de les spécifier lors de l'exécution.
FileInputFormat.addInputPath(job, new Path(ourArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(ourArgs[1]));
```

Exemple – Occurences de mots


```
        // On lance la tâche Hadoop. Si elle s'est effectuée
        // correctement, on renvoie 0. Sinon, on renvoie -1.
        if(job.waitForCompletion(true))
            System.exit(0);
        System.exit(-1);
    }
```

Exemple – Occurences de mots

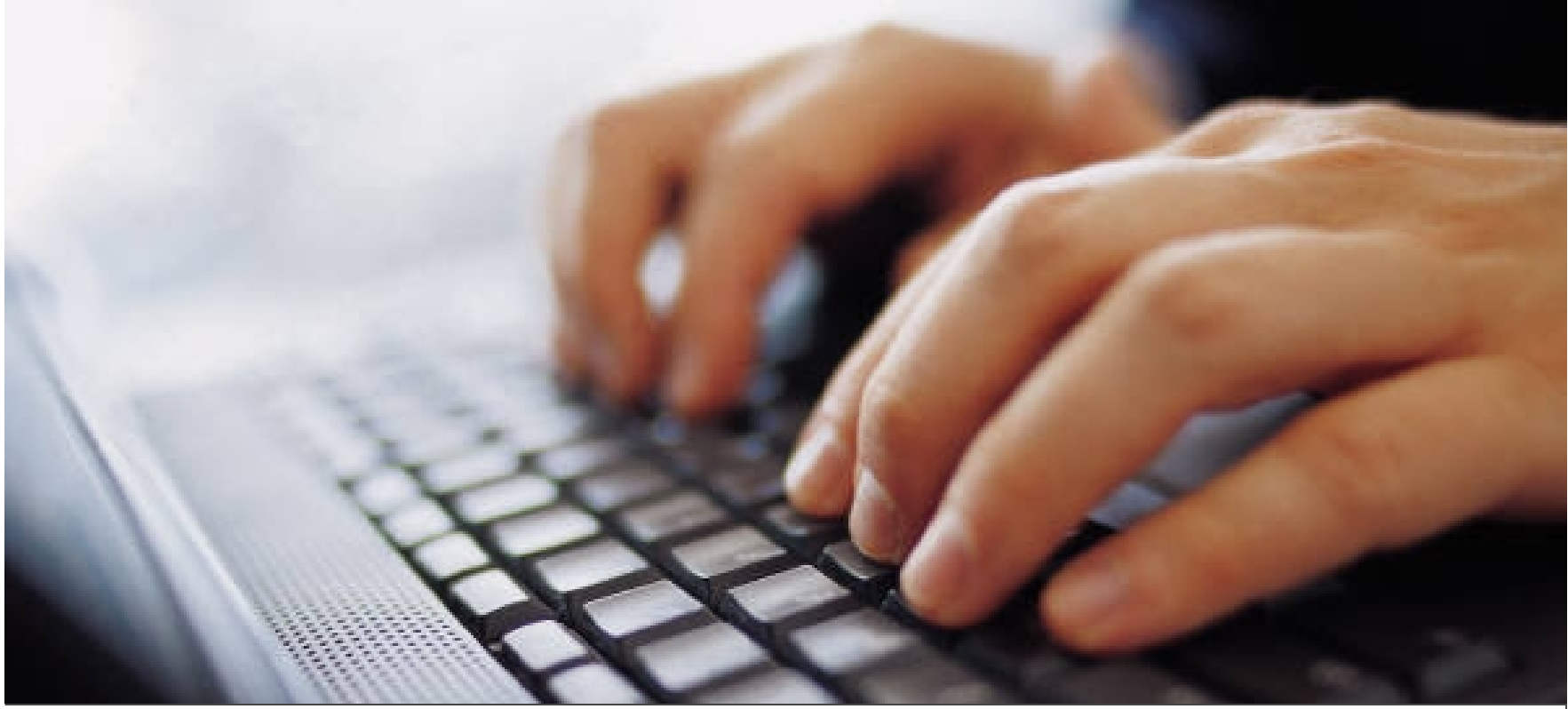

```
private static final IntWritable ONE=new IntWritable(1);

// La fonction MAP elle-même.
protected void map(Object offset, Text value, Context context)
                    throws IOException, InterruptedException
{
    // Un StringTokenizer va nous permettre de parcourir chacun des
    // mots de la ligne qui est passée à notre opération MAP.
    StringTokenizer tok=new StringTokenizer(value.toString(), " ");

    while(tok.hasMoreTokens())
    {
        Text word=new Text(tok.nextToken());
        // On renvoie notre couple (clef;valeur): le mot courant suivi
        // de la valeur 1 (définie dans la constante ONE).
        context.write(word, ONE);
    }
}
```

Exemple – Occurences de mots


```
    // La fonction REDUCE elle-même. Les arguments: la clef key, un
    // Iterable de toutes les valeurs qui sont associées à la clef en
    // question, et le contexte Hadoop (un handle qui nous permet de
    // renvoyer le résultat à Hadoop).
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context)
                       throws IOException, InterruptedException
    {
        // Pour parcourir toutes les valeurs associées à la clef fournie.
        Iterator<IntWritable> i=values.iterator();
        int count=0;  // Notre total pour le mot concerné.
        while(i.hasNext())   // Pour chaque valeur...
            count+=i.next().get();    // ...on l'ajoute au total.
        // On renvoie le couple (clef;valeur) constitué de notre clef key
        // et du total, au format Text.
        context.write(key, new Text(count+" occurences."));
    }
}
```



5 Hadoop: utilisation (2)

Exemple 2 – Anagrammes / Driver


```
// Le main du programme.
public static void main(String[] args) throws Exception
{
    // Créé un object de configuration Hadoop.
    Configuration conf=new Configuration();

    // Permet à Hadoop de lire ses arguments génériques,
     // récupère les arguments restants dans ourArgs.
    String[] ourArgs=new GenericOptionsParser(conf,
                          args).getRemainingArgs();

    // Obtient un nouvel objet Job: une tâche Hadoop. On
   // fourni la configuration Hadoop ainsi qu'une description
    // textuelle de la tâche.
    Job job=Job.getInstance(conf, "Anagrammes v1.0");
```

Exemple 2 – Anagrammes / Driver


```
        job.setJarByClass(Anagrammes.class);
        job.setMapperClass(AnagrammesMap.class);
        job.setReducerClass(AnagrammesReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Défini les fichiers d'entrée du programme et le
         // répertoire des résultats, depuis la ligne de commande.
        FileInputFormat.addInputPath(job, new Path(ourArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(ourArgs[1]));

        // On lance la tâche Hadoop. Si elle s'est effectuée
        // correctement, on renvoie 0. Sinon, on renvoie -1.
        if(job.waitForCompletion(true))
            System.exit(0);
        System.exit(-1);
    }
```

Exemple 2 – Anagrammes / Mapper


```
// Classe MAP
public class AnagrammesMap extends Mapper<Object, Text, Text, Text>
{
    // La fonction MAP elle-même.
    protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException
    {
        // Ordonne les lettres de la valeur d'entrée par ordre alphabétique.
        // Il s'agira de la clef de sortie.
        char[] letters=value.toString().toLowerCase().toCharArray();
        Arrays.sort(letters);

        // On renvoie le couple (clef;valeur), en convertissant la clef
        //au type Hadoop (Text).
        context.write(new Text(new String(letters)), value);
    }
}
```

Exemple 2 – Anagrammes / Reducer


```
// La classe REDUCE.
public class AnagrammesReduce extends Reducer<Text, Text, Text, Text>
{
  public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException
    {
        // Pour parcourir toutes les valeurs associées à la clef fournie.
        Iterator<Text> i=values.iterator();
        String result="";
        Boolean first=true;
        while(i.hasNext())   // Pour chaque valeur...
        {
            if(first)   // Premier mot, donc on n'inclut pas le symbole "|".
            {
                // Notre chaîne de résultat contient initialement
                // le premier mot.
                result=i.next().toString();
                first=false;
            }
```

Exemple 2 – Anagrammes / Reducer


```
            // Pas le premier: on concatene le mot à la chaîne de resultat.
            else
                result=result+" | "+i.next().toString();
        }

        // On renvoie le couple (clef;valeur) constitué de
        // notre clef key et de la chaîne concaténée.
        context.write(key, new Text(result));
    }
}
```

Compilation

- **Un programme map/reduce Hadoop se compile comme tout programme Java « traditionnel ».**

- **Il faut ensuite le compresser au sein d'un paquet Java .jar classique, en faisant par exemple:**

```
mkdir -p org/unice/hadoop/wordcount
mv *.class org/unice/hadoop/wordcount
jar -cvf unice_wcount.jar -C . org
```

- **Avantage du .jar: on peut tout à fait y inclure plusieurs programmes map/reduce différent, et sélectionner celui qu'on souhaite à l'exécution.**

Exécution

- **Là encore, on utilise le client console `hadoop` pour l'exécution.**
  Synopsis:

```
hadoop jar [JAR FILE] [DRIVER CLASS] [PARAMETERS]
```

Par exemple:

```
hadoop jar wcount_unice.jar org.unice.hadoop.wordcount.WCount \
           /input/poeme.txt /results
```

Résultats

- **Hadoop stocke les résultats dans une série de fichiers dont le nom respecte le format:**

  part-r-XXXX

  … avec XXXX un compteur numérique incrémental.

- **On a un fichier part-r par opération Reduce exécutée. Le « r » désigne le résultat de l'opération Reduce. On peut également demander la génération de la sortie des opérations map – dans des fichiers part-m-*.**

- **En cas de succès, un fichier vide _SUCCESS est également créé.**

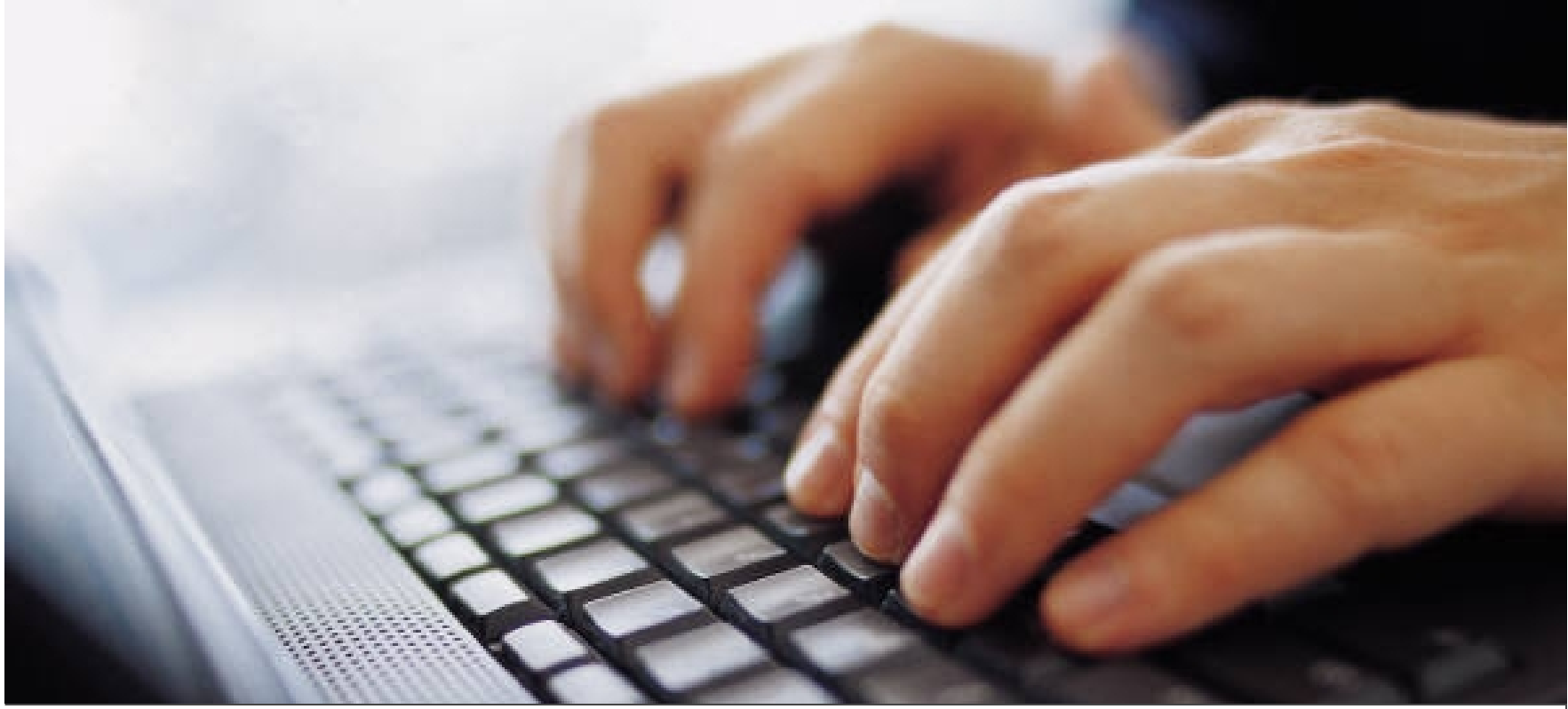
Exécution

- **Là encore, on utilise le client console `hadoop` pour l'exécution.**
  Synopsis:

```
hadoop jar [JAR FILE] [DRIVER CLASS] [PARAMETERS]
```

Par exemple:

```
hadoop jar wcount_unice.jar org.unice.hadoop.wordcount.WCount \
           /input/poeme.txt /results
```



6 Autres langages; autres outils Hadoop

Autres langages de programmation

- **Pour développer des programmes map/reduce Hadoop dans d'autres langages: utilitaire Streaming, distribué avec Hadoop.**

- **Il s'agit en réalité d'une application Hadoop Java classique capable d'invoquer un intépréteur/un binaire sur tous les nœuds du cluster.**

- **On spécifie à l'utilitaire deux arguments: le programme à exécuter pour l'opération map, et celui à exécuter pour l'opération reduce.**

Autres langages de programmation

- **Synopsis:**

```
hadoop jar hadoop-streaming-X.Y.Z.jar -input [HDFS INPUT FILES] \
                                      -output [HDFS OUTPUT FILES] \
                                      -mapper [MAP PROGRAM]  \
                                      -reducer [REDUCE PROGRAM]
```

- **Par exemple:**

```
hadoop jar hadoop-streaming-X.Y.Z.jar -input /poeme.txt \
           -output /results -mapper ./map.py -reducer ./reduce.py
```

Format des données

- **Pour passer les données d'entrée et recevoir les données de sortie, streaming fait appel aux flux standards respectifs *stdin* et *stdout*.**

- **Le format respecte le schéma suivant:**

```
CLEF[TABULATION]VALEUR
```

  … avec un couple (clef;valeur) par ligne.

- **Dans le programme reduce, on est susceptible de recevoir plusieurs groupes, correspondant à plusieurs clefs distinctes, en entrée.**

Exemple – Occurences de mot, Python

- **Opération map:**

```
import sys

# Pour chaque ligne d'entrée.
for line in sys.stdin:
  # Supprimer les espaces autour de la ligne.
  line=line.strip()
  # Pour chaque mot de la ligne.
  words=line.split()
  for word in words:
    # Renvoyer couple clef;valeur: le mot comme clef, l'entier "1" comme
    # valeur.
    # On renvoie chaque couple sur une ligne, avec une tabulation entre
    # la clef et la valeur.
    print "%s\t%d" % (word, 1)
```

Exemple – Occurences de mot, Python

- **Opération reduce:**

```
import sys

total=0;  # Notre total pour le mot courant.
# Contient le dernier mot rencontré.
lastword=None

# Pour chaque ligne d'entrée.
for line in sys.stdin:
  # Supprimer les espaces autour de la ligne.
  line=line.strip()

  # Récupérer la clef et la valeur, convertir la valeur en int.
  word, count=line.split('\t', 1)
  count=int(count)
```

Exemple – Occurences de mot, Python

- **Opération reduce (suite):**

```
  # On change de mot (test nécessaire parce qu'on est susceptible d'avoir
  # en entrée plusieurs clefs distinctes différentes pour une seule et
  # même exécution du programme - Hadoop triera les couples par clef
  # distincte).
  if word!=lastword and lastword!=None:
    print "%s\t%d occurences" % (lastword, total)
    total=0;
  lastword=word

  total=total+count  # Ajouter la valeur au total

# Ecrire le dernier couple (clef;valeur).
print "%s\t%d occurences" % (lastword, total)
```

Exemple – Occurences de mot, Bash

- **Opération map:**

```
# Pour chaque ligne d'entrée.
while read line; do
  # Supprimer les espaces autour de la ligne.
  line=$(echo "$line"|sed 's/^\s*\(.\+\)\s*$/\1/')
  # Pour chaque mot de la ligne.
  for word in $line; do
    # Renvoyer couple clef;valeur: le mot comme clef, l'entier "1" comme
    # valeur.
    # On renvoie chaque couple sur une ligne, avec une tabulation entre
    # la clef et la valeur.
    echo -e $word"\t"1
  done
done
```

Exemple – Occurences de mot, Bash

- **Opération reduce:**

```
lastword=""
total=0

# Pour chaque ligne d'entrée.
while read line; do
  # Supprimer les espaces autour de la ligne.
  line=$(echo "$line"|sed 's/^\s*\(.\+\)\s*$/\1/')

  # Recuperer mot et occurrence (IE, clef et valeur)
  word=$(echo "$line"|awk -F "\t" '{print $1}');
  nb=$(echo "$line"|awk -F "\t" '{print $2}');
```

Exemple – Occurences de mot, Bash

- **Opération reduce (suite):**

```
  # On change de mot (test nécessaire parce qu'on est susceptible d'avoir
  # en entrée plusieurs clefs distinctes différentes pour une seule et
  # même exécution du programme - Hadoop triera les couples par clef
  # distincte).
  if [ "$word" != "$lastword" ] && [ "$lastword" != "" ]; then
    echo -e $lastword"\t"$total" occurences."
    total=0;
  fi
  lastword="$word"

  total=$[ $total + $nb ]   # Ajouter la valeur au total.
done

# Ecrire le dernier couple (clef;valeur).
echo -e $lastword"\t"$total" occurences."
```

Autres outils associés à Hadoop

- **Nombreux *front-ends* graphiques (projet Hue…).**
- **Apache Hive: outil d'analyse/recoupage/etc. de larges volumes de données, associé à un langage similaire à SQL: HiveQL.**
- **Apache Pig: outil permettant le développement de programmes map/reduce dans un langage simplifié et accessible (Piglatin).**
- **Sqoop: outil d'importation/exportation de données entre HDFS et de nombreux systèmes de gestion de bases de données classiques.**

… et beaucoup d'autres …

Conclusion

- **Le présent cours constitue une introduction. L'API Hadoop, et le *framework* en général, présentent de nombreuses autres possibilités.**

- **Se référer au guide de lectures, ou encore à la documentation Hadoop, pour approfondir le sujet.**

- **Expérimenter ! Se familiariser avec l'approche map/reduce et l'utilisation de Hadoop nécessite de la pratique.**