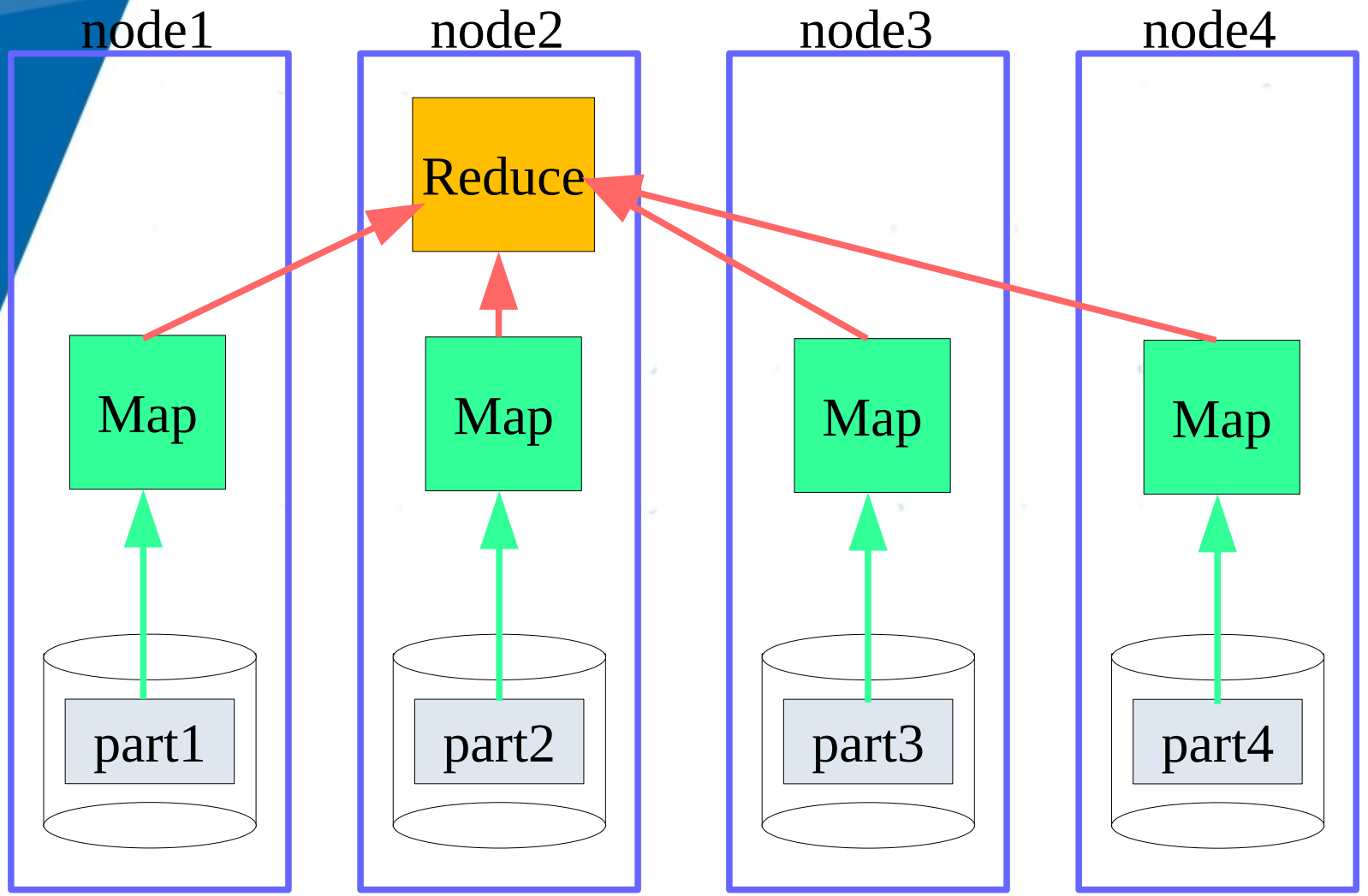
Projet Hagidoop

ENSEEIHT, 2023

Daniel Hagimont
daniel.hagimont@irit.fr

Principe du map-reduce



Exemple d'application comptage de mots

- En itératif

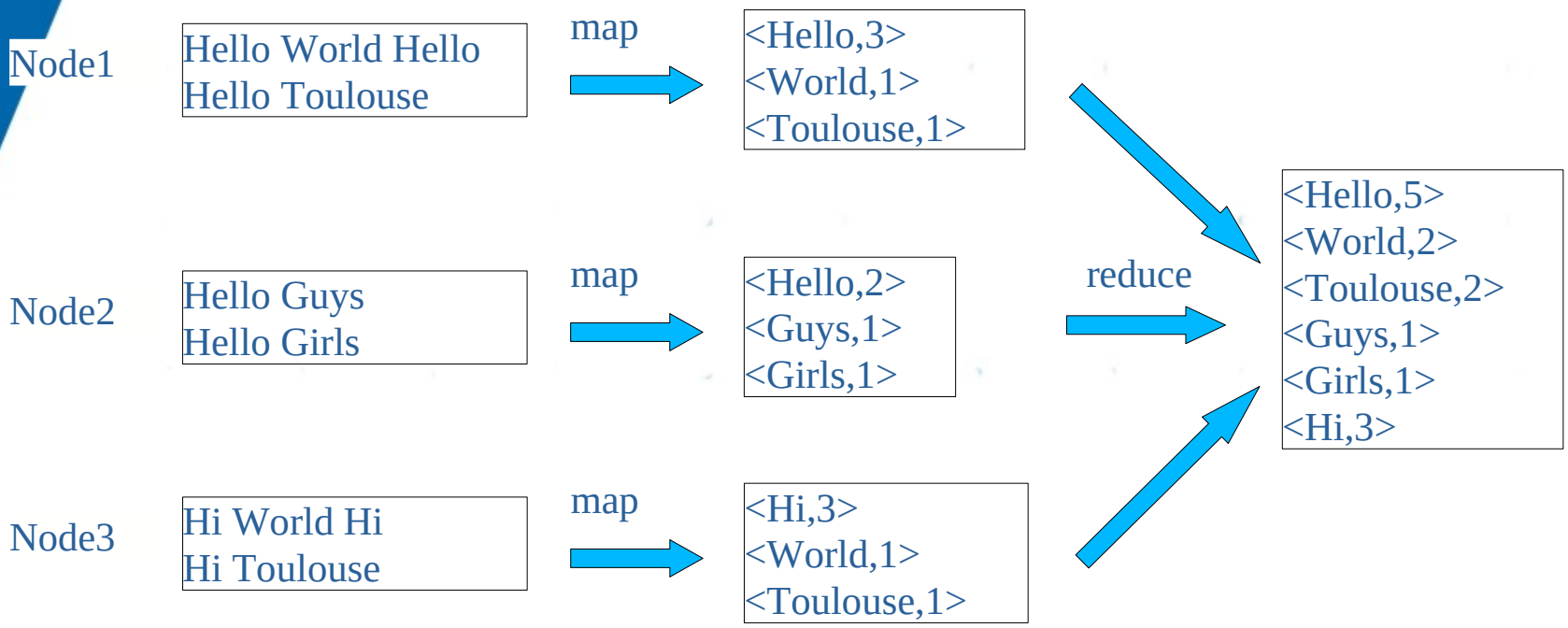
```
HashMap<String,Integer> hm = new HashMap<String,Integer>();

// ouvrir fichier à lire : lnr
while (true) {
        String l = lnr.readLine();
        if (l == null) break;
        String tokens[] = l.split(" ");
        for (String tok : tokens) {
                if (hm.containsKey(tok))
                        hm.put(tok, hm.get(tok)+1);
                else
                        hm.put(tok, 1);
        }
}

// recopier la hashmap dans le fichier résultat
```

Exemple d'application comptage de mots

- En map-reduce



Exemple d'application comptage de mots

- En map-reduce

Read de KV (<xxx,ligne> pour wordcount)

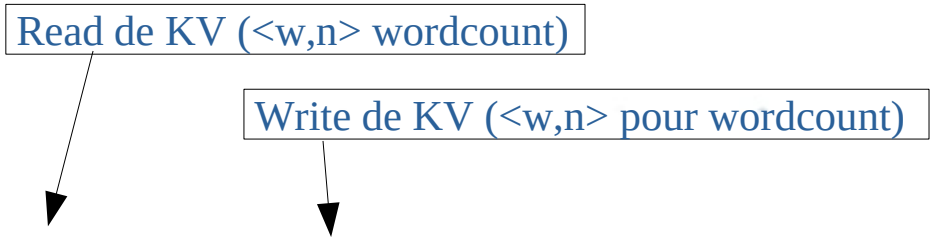
Write de KV (<w,n> pour wordcount)

```
public void map(Reader reader, Writer writer) {

        HashMap<String,Integer> hm = new HashMap<String,Integer>();
        KV kv;
        while ((kv = reader.read()) != null) {
                String tokens[] = kv.v.split(" ");
                for (String tok : tokens) {
                        if (hm.containsKey(tok))
                                hm.put(tok, hm.get(tok)+1);
                        else
                                hm.put(tok, 1);
                }
        }
        for (String k : hm.keySet())
                writer.write(new KV(k,hm.get(k).toString()));
        }
}
```

Exemple d'application comptage de mots

- En map-reduce



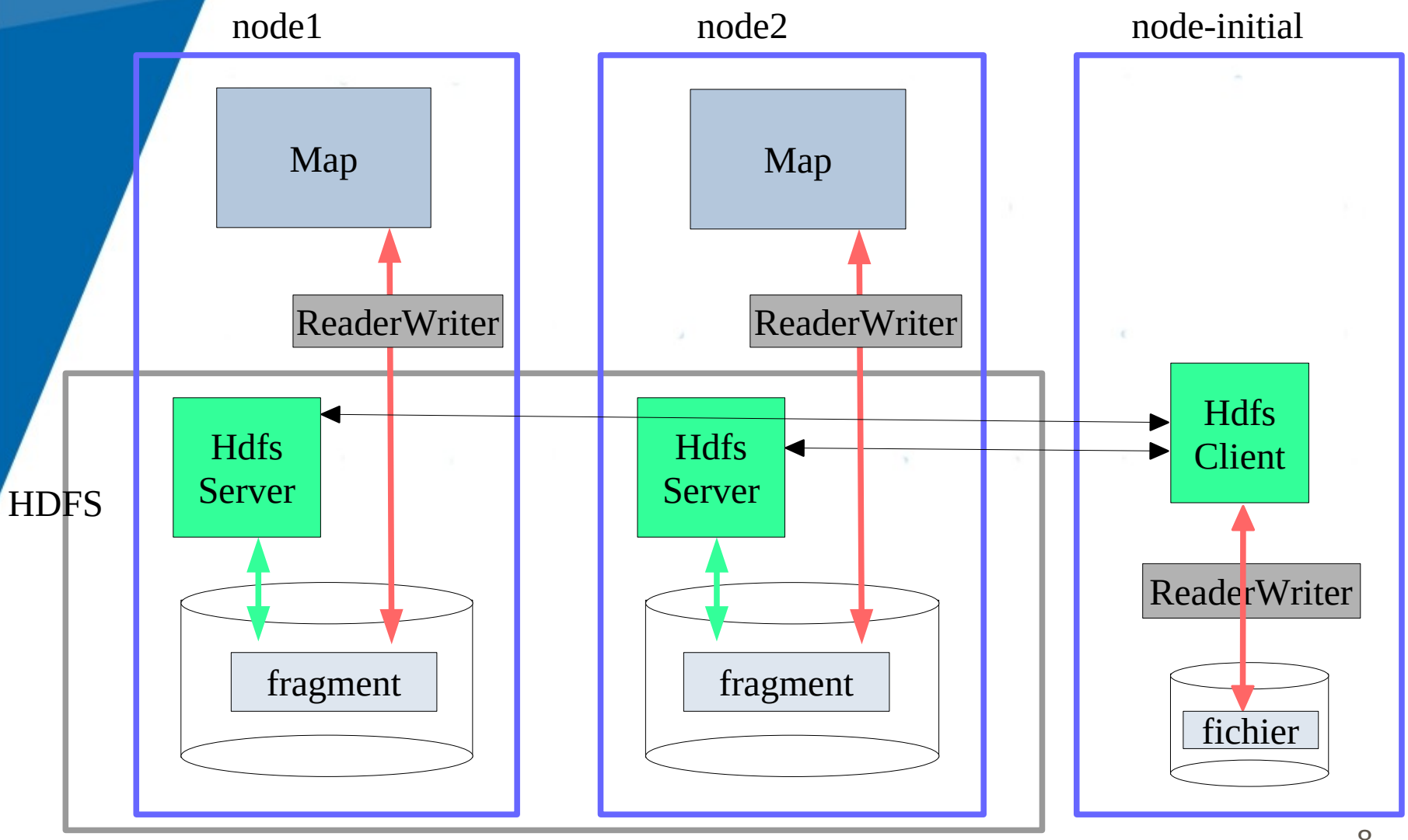

```
public void reduce(Reader reader, Writer writer) {

        HashMap<String,Integer> hm = new HashMap<String,Integer>();
        KV kv;
        while ((kv = reader.read()) != null) {
                if (hm.containsKey(kv.k))
                        hm.put(kv.k, hm.get(kv.k)+Integer.parseInt(kv.v));
                else
                        hm.put(kv.k, Integer.parseInt(kv.v));
        }
        for (String k : hm.keySet())
                writer.write(new KV(k,hm.get(k).toString()));
}
```

Hagidoop

- Système permettant d'exécuter ces applications dans un cluster
  - HDFS : Hagidoop Distributed File System
    - Permet de stockage des données à traiter sous forme de fragments sur les noeuds du cluster
  - Hagidoop : exécution des applications MapReduce
    - Permet l'exécution des Map en parallèle et du Reduce

Architecture HDFS



HDFS

- Permet de gérer des fichiers fragmentés sur les nœuds
  - Quand on copie un fichier du FS local dans le FS HDFS, le fichier est coupé en fragments qui sont copiés sur les nœuds.
  - Quand on copie un fichier du FS HDFS dans le FS local, les fragments sont rassemblés pour obtenir le fichier complet sur le FS local.
- Les fragments sont copiés sur le FS local du nœud avec un nom particulier
- Les fragments sont de taille variable (en fonction du nombre de nœuds) et non répliqués
- Les ReaderWriter permettent des lectures/écritures cohérentes (par exemple pour ne pas couper au milieu d'une ligne ou d'un mot)

HDFS

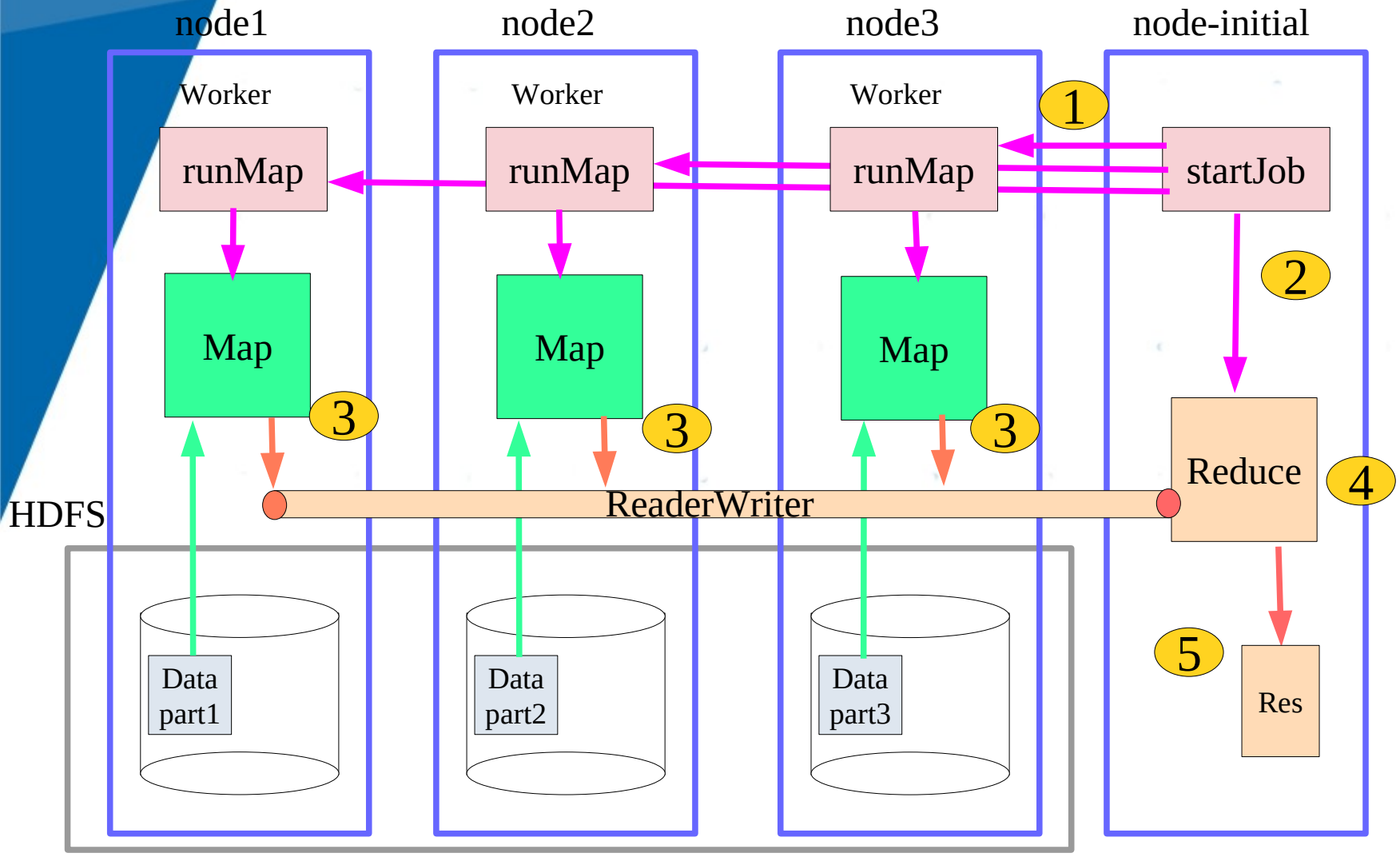
- Utilisation externe (depuis un shell)

```
public class HdfsClient {
        public static void HdfsDelete(String fname) {…}
        public static void HdfsWrite(int fmt, String fname) {…}
        public static void HdfsRead(String fname) {…}

        public static void main(String[] args) {
                // java HdfsClient <read|write> <txt|kv> <file>
        }
}
```

- Communique avec HdfsServer avec TCP

Architecture de Hagidoop



Modèle de programmation

```
public interface Map extends Serializable {
        public void map(Reader reader, Writer writer);
}

public interface Reduce extends Serializable {
        public void reduce(Reader reader, Writer writer);
}

public interface MapReduce extends Map, Reduce {
}

public class JobLauncher {
        public static void startJob (MapReduce mr, int format, String fname) {
        …
        }
}
```

Hagidoop

- Lancement d'un Job depuis une application (noeud initial)

```
public class JobLauncher {

        public static void startJob (MapReduce mr, int format, String fname) {
        …
        }
}
```

- Interface (RMI) du démon appelé Worker

```
public interface Worker extends Remote {
        public void runMap (
                Map m,
                FileReaderWriter reader,
                NetworkReaderWriter writer)
        throws RemoteException;
}
```

hagidoop

1. startJob lance les map en appelant runMap sur les Worker (en leur donnant un map, reader et writer)
2. startJob lance le reduce qui récupère les résultats des map et les traite
3. Les map calculent en lisant localement un fragment (avec reader) et génèrent des données (KV), envoyées au reduce (avec writer)
4. Le reduce traite les données qu'il reçoit (avec reader) au fur et à mesure, en parallèle des map
5. Quand tous les map se terminent le Reduce copie le résultat final dans un fichier local (avec writer)

Les ReaderWriter

- Pour les lectures/écriture dans Hagidoop
- Repose sur des KV

```
public class KV implements Serializable, Cloneable {
        public static final String SEPARATOR = "<->";
        public String k;
        public String v;

        public KV() {}

        public KV(String k, String v) {
                super();
                this.k = k;
                this.v = v;
        }
}
```

Les ReaderWriter

- Pour implanter des objets de communication

```
public interface Reader {
        public KV read();
}
public interface Writer {
        public void write(KV record);
}

public interface ReaderWriter extends Reader, Writer, Serializable {}
```

Les ReaderWriter

- Pour lire/écrire dans un fichier
  - On gère deux formats (donc deux classes qui implantent ces formats) :
    - TxtFile : une classe pour les fichiers texte
    - KVFile : une classe pour des fichiers KV

```
public interface FileReaderWriter extends ReaderWriter {
        public static final int FMT_TXT = 0;
        public static final int FMT_KV = 1;
        public void open(String mode);
        public void close();
        public long getIndex();
        public String getFname();
        public void setFname(String fname);
}
```

Les ReaderWriter

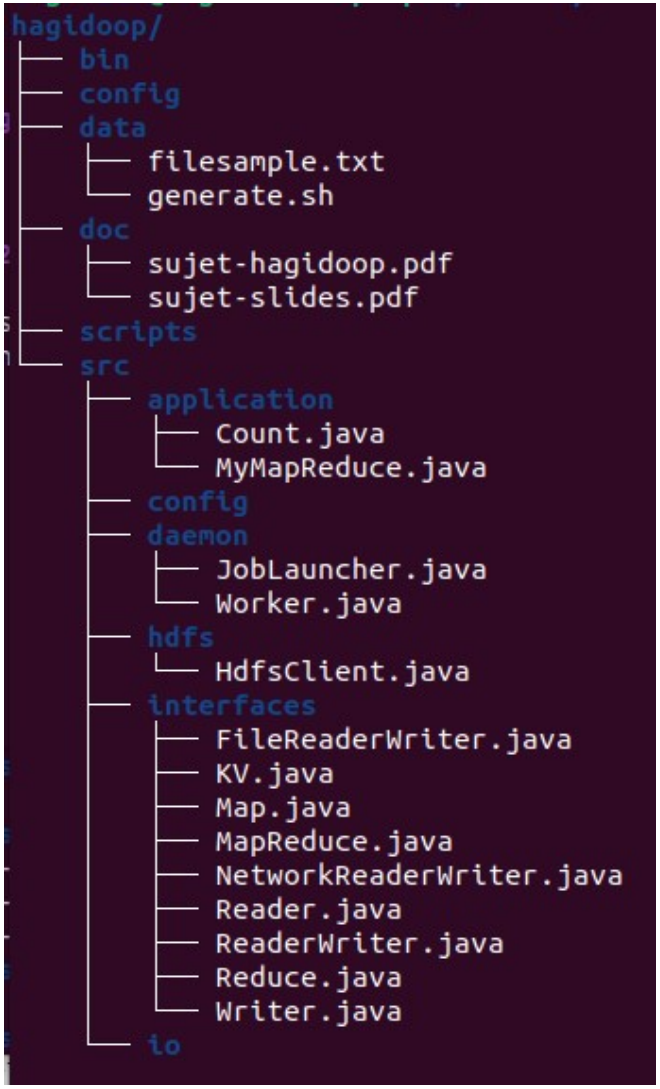
- Pour lire/écrire à distance entre Maps et Reduce
  - Chaque map peut faire openClient()
  - Le startJob peut faire openServer(), puis accept() pour recevoir une connexion de chaque map
  - Le startJob doit lire sur ces connexions et envoyer tout ça au reduce

```
public interface NetworkReaderWriter extends ReaderWriter {
        public void openServer();
        public void openClient();
        public NetworkReaderWriter accept();
        public void closeServer();
        public void closeClient();
}
```

Interfaces

- Vous devez respecter les interfaces/classes :
  - KV
  - Reader, Writer, ReaderWriter
  - FileReaderWriter, NetworkReaderWriter
  - Map, Reduce, MapReduce
  - JobLauncher, HdfsClient
- Le reste est libre

Architecture



A faire

- Implanter Hagidoop
- Faire des outils de déploiement (local et distribué)
- Évaluer la scalability
  - Pour un gros fichier, j'augmente le nombre de noeuds et ça diminue le temps d'exécution
- Potentiellement, si le reduce est un bottleneck, évoluer vers plusieurs reduce.

Suivi

- Envoyer constitution binome : avant le 24/11/23 18h
  - Pas de contrainte sur les groupes
  - Mail à daniel.hagimont@irit.fr
    - Binôme : Alfred Durant (L1) / Marcel Dupont (L2)
    - Pas de mail : 0/20
- Suivis
  - Armel Jeatsa et Daniel Hagimont
  - Pas obligatoire
    - démarrez tôt sinon sortie de route
    - Les premières séances (explications) sont déterminantes
  - Les séances
    - 1/12/23
    - 12-13/12/23
    - 20/12/23 (en visio)
    - 11/01/24
    - Oral : 16/01/24