MapReduce et Hadoop

Alexandre Denis – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest
France

ENSEIRB PG306

Fouille de données – *Big Data*

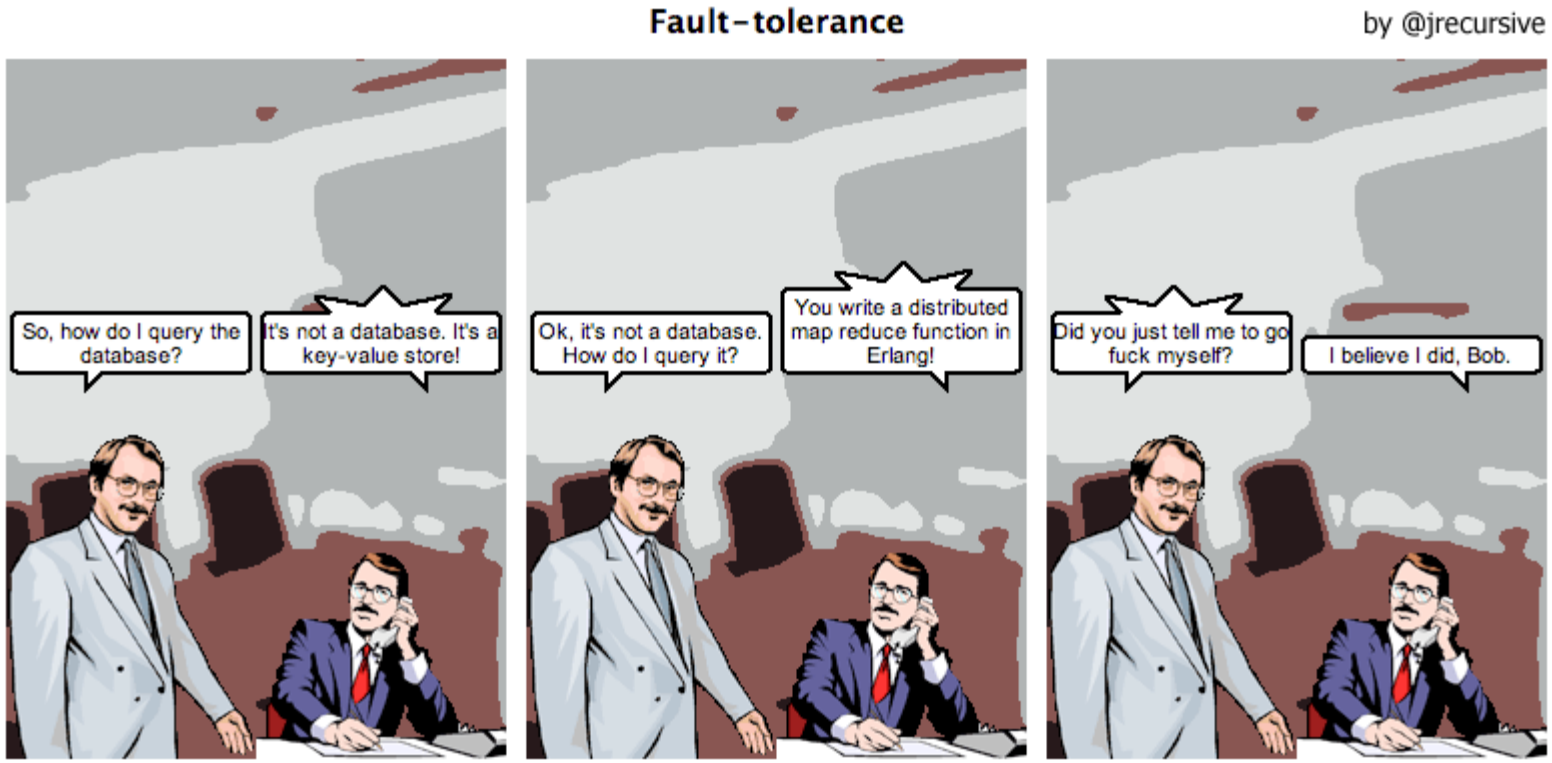
- Recherche & indexation de gros volumes
- Appliquer une opération simple à beaucoup de données
  - Compter les mots
  - Rechercher un mot
  - Compter les occurrences d'un pattern
  - Rechercher les liens pointant vers une page
- Opérations limitées par l'**accès aux données**
  - Pas par la puissance de calcul
- Grosses bases de données, moteur de recherche internet, logs d'un serveur
  - ex.: Google, Yahoo, Facebook

Map et reduce

- **Opérations fonctionnelles** d'ordre supérieur
- Map: (f, [a, b, c, ...]) -> [f(a), f(b), f(c), ...]
  - Applique une fonction à tous les éléments d'une liste
  - ex.: map((f: x->x + 1), [1, 2, 3]) = [2, 3, 4]
  - Intrinsèquement **parallèle**
- Reduce: (f, [a, b, c, ...]) -> f(a, f(b, f(c, ...)))
  - Applique une fonction récursivement à une liste
  - = fold en langages fonctionnels, ~MPI_Reduce
- Purement fonctionnel
  - Pas de variables globales, pas d'effets de bord

Modèle de données <k, v>

- Modèle <key, value>

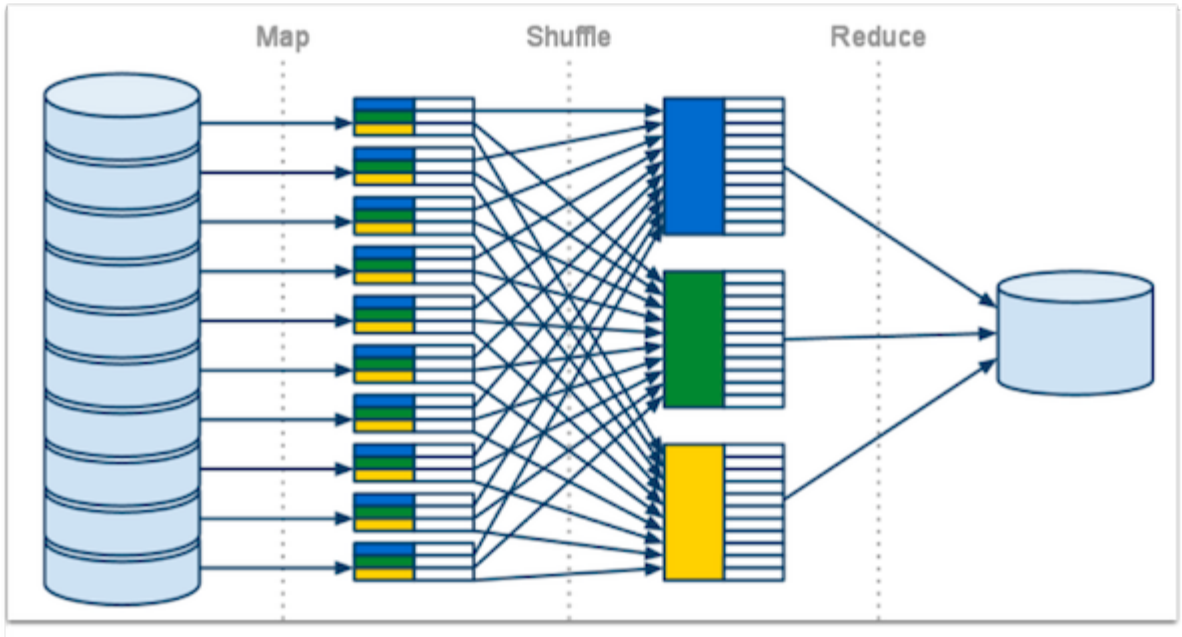


Modèle de données <k, v>

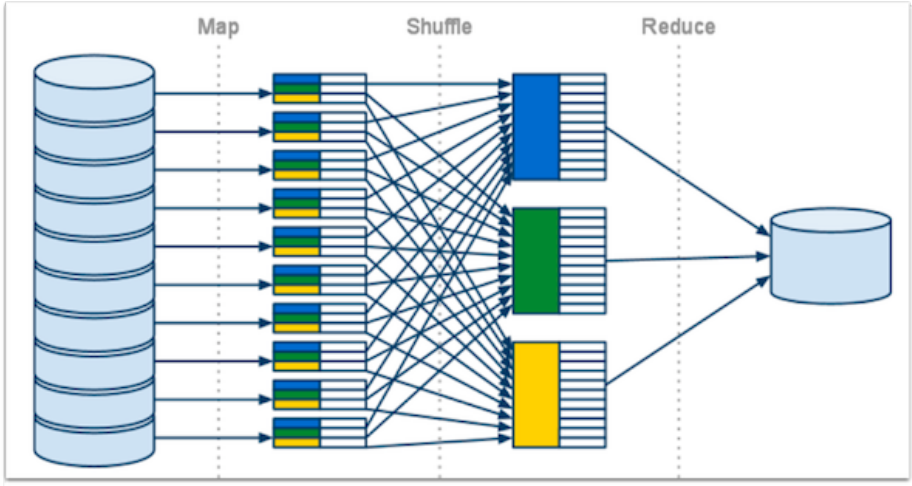
- Modèle <key, value>
  - Tableau associatif (i.e. ~hashtable)
  - Modèle simple de base de données
    - = data store
- MapReduce opère sur des **couples <k, v>**
  - En entrée, en sortie, en interne
  - Pas nécessairement dans le même espace de nommage à chaque étape

MapReduce

- Modèle de programmation MapReduce
  - Map, Shuffle, Reduce



MapReduce



- **Map**
  - <k, v> -> <k2, v2>
  - Application d'une fonction sur chaque donnée de départ
  - k et k2 : pas nécessairement dans le même espace, ni le même nombre
- **Shuffle**
  - Tri des résultats par clef
  - i.e. on regroupe les v2 qui ont la même k2
- **Reduce**
  - Application d'une fonction de réduction sur tous les groupes <k2, v2>

MapReduce - exemple

- Compter les occurrences des mots d'une collection de documents
- Entrée : < nom, contenu >
- Map : pour chaque mot du contenu -> <mot, 1>
- Shuffle : regroupe tous les <mot, 1> pour un mot donné
- Reduce : <mot, nombre> -> <mot, total>
- Sortie : collection de <mot, occurrences>

MapReduce – exemple 2

- Déterminer les documents pointant vers une URL
  - Résolution inverse de liens
- Entrée : collection de < URL, contenu >
- Map : pour chaque lien du contenu -> < URL lien, URL source>
- Shuffle : regroupe tous les <URL lien, URL source> pour une URL donnée
- Reduce : <URL lien, URL source> -> <URL lien, liste de sources>
- Sortie : collection de <URL, liste de sources>

Apache Hadoop

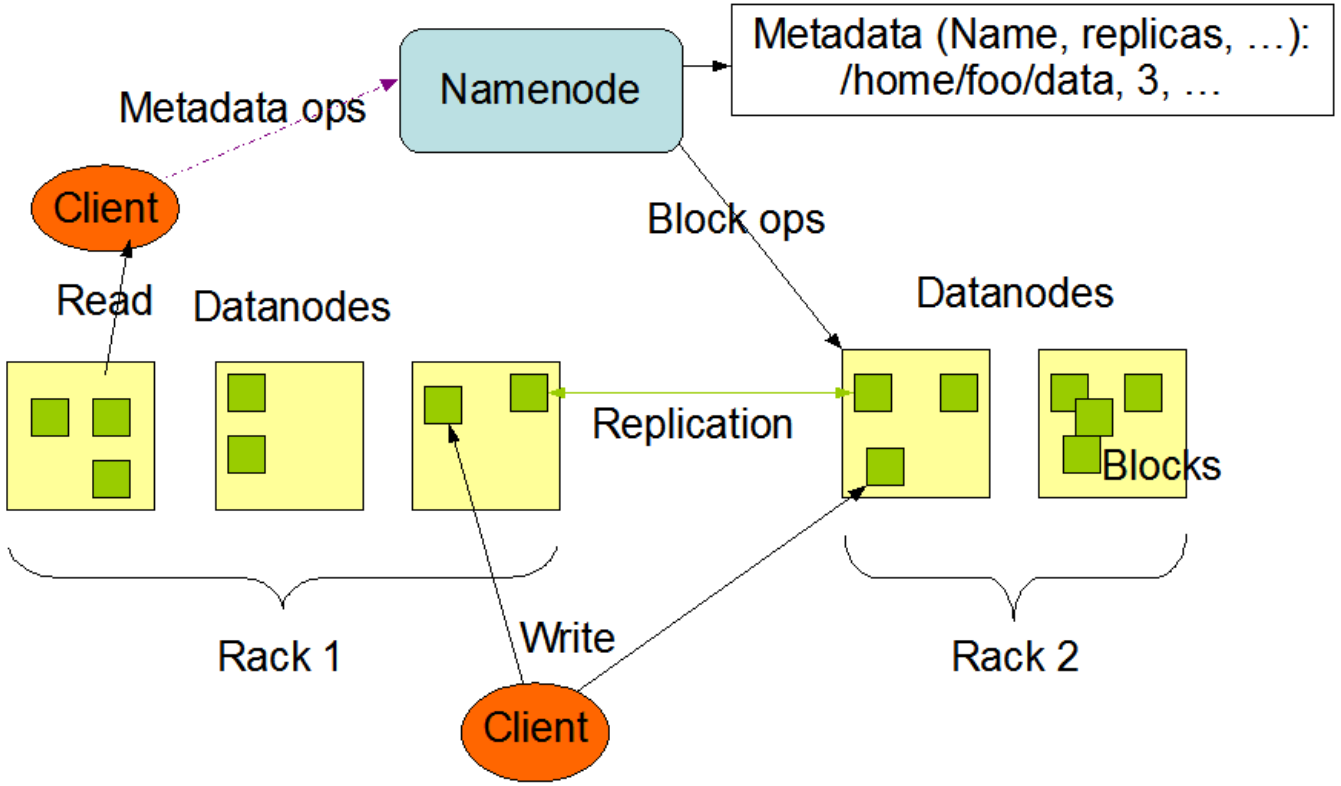
- Framework de la fondation Apache
- Issu de travaux conjoints de Google et Apache Lucene (Yahoo)
  - ~ 2004
- Pensé pour les gros volumes – **Big Data**
  - Pétaoctets de données
  - Milliers de noeuds de stockage/calcul
- Implémentation en Java
- Contient : MapReduce, HDFS, Hbase, etc.

HDFS - stockage

- HDFS – Hadoop Distributed File System
  - Système de fichier distribué
  - Scalabilité, tolérance aux fautes
  - Modèle de cohérence simple
    - Write-once, read-many
  - Informations de localité
    - Amener les calculs aux données est moins coûteux que d'amener les données aux noeuds de calcul
-

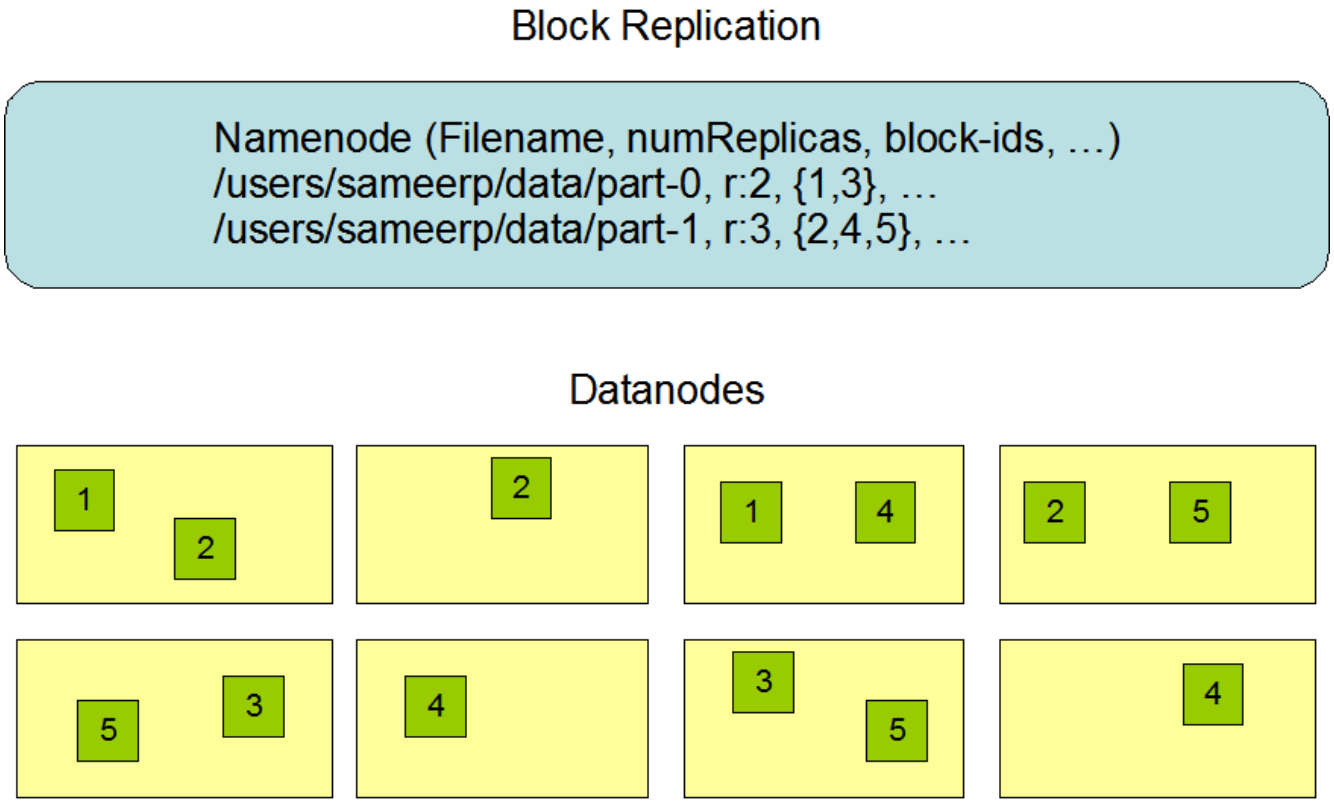
HDFS - architecture

- NameNode- master, maintient la hierarchie de fichiers
- DataNode- slave, un par noeud, gère l'accès disque des données



HDFS - replication

- HDFS réplique les blocs de données pour la tolérance aux pannes et la performance



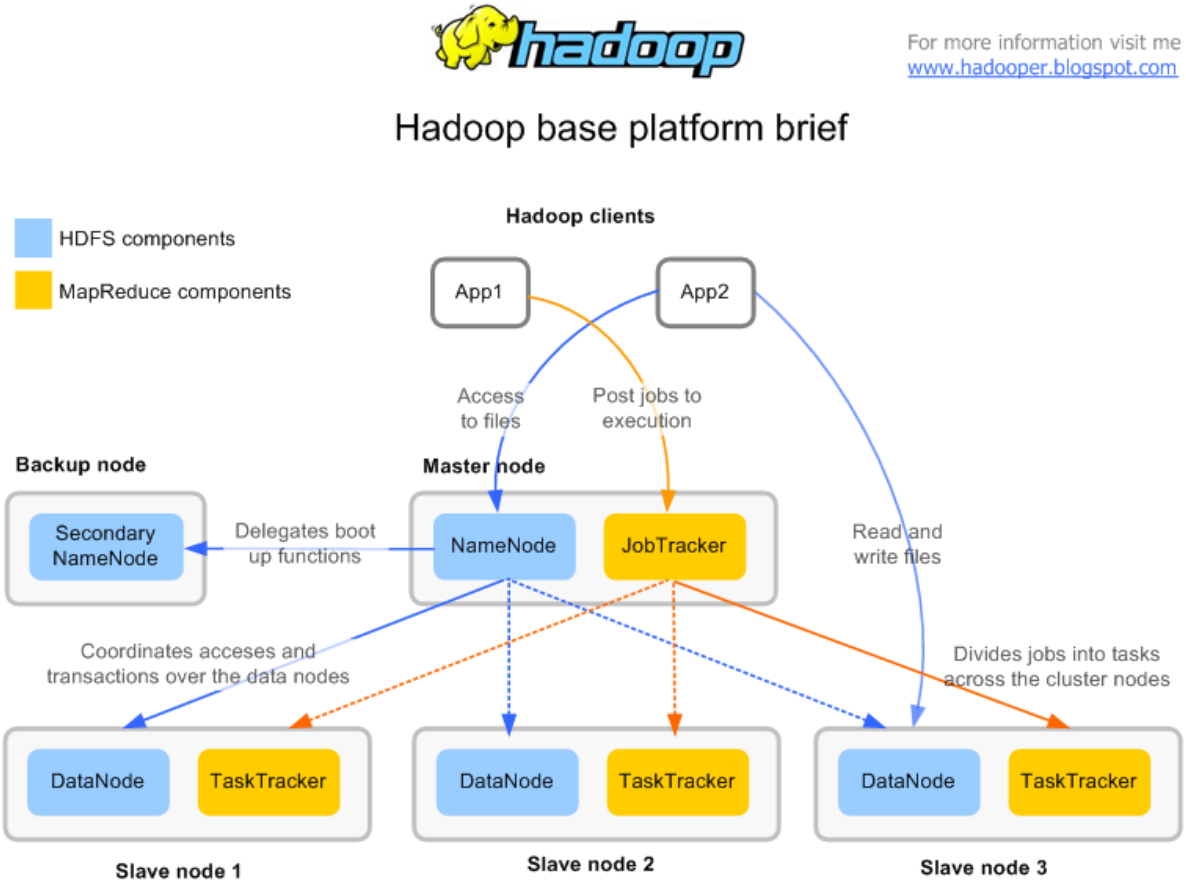
HDFS - commandes

- API native Java
- Interface ligne de commande
  - Interface similaire au shell classique
  - Accès à l'aide de : `hadoop dfs <commande>`

Action	Command
Create a directory named /foodir	bin/hadoop dfs -mkdir /foodir
Remove a directory named /foodir	bin/hadoop dfs -rmr /foodir
View the contents of a file named /foodir/myfile.txt	bin/hadoop dfs -cat /foodir/myfile.txt

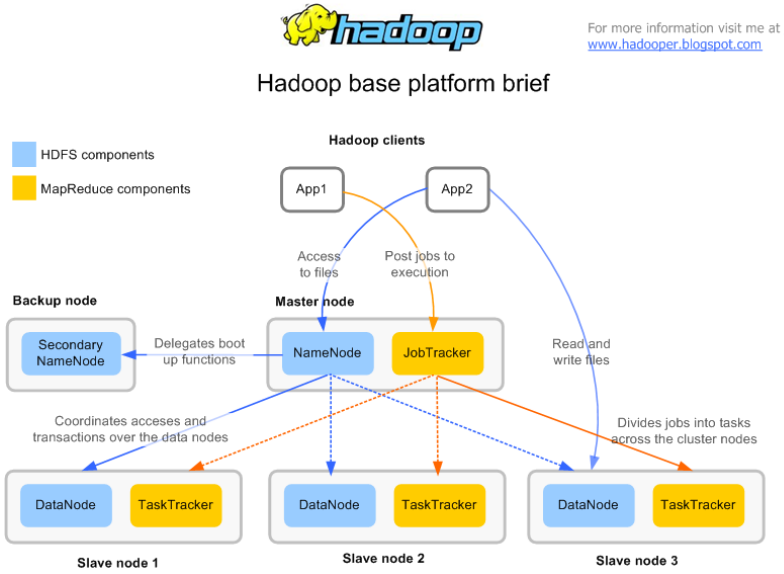
Hadoop MapReduce – architecture

- Double arboressence : données (HDFS) et tâches (MapReduce)



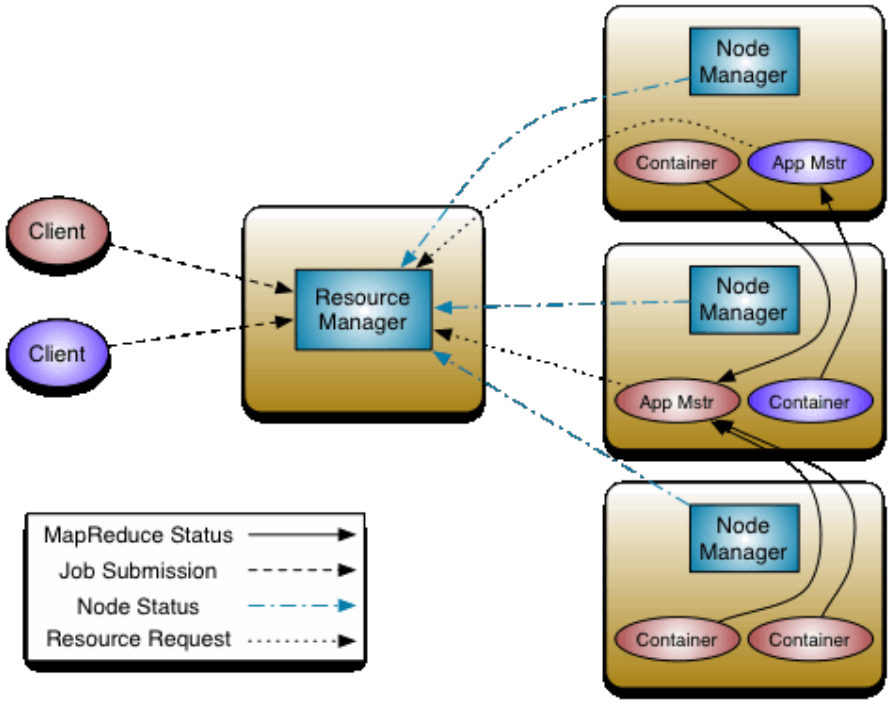
Hadoop MapReduce – déroulement

- Envoi des calculs près des données
  - Le Client soumet un job au **JobTracker**
  - Le **JobTracker** interroge le NameNode pour localiser les données
  - Le **JobTracker** localise des **TaskTracker** près des données
  - Le **JobTracker** soumet des tâches aux **TaskTrackers**
  - Le **TaskTracker** notifie la terminaison au **JobTracker**
  - Le Client polle le **JobTracker** pour avoir l'état de son job



YARN – moteur d'exécution MapReduce

- v2.0 du moteur d'exécution MapReduce
  - Séparation entre Resource Manager (global) Node Manager (monitoring)
  - L'Application Master (déroule la DAG) est une tâche comme une autre



Placement et ordonnancement

- Un Job est un **dataflow**
  - Map, shuffle, et reduce sont des tâches
  - Le job est exprimé sous forme de DAG de tâches
- Placement
  - Si possible, les maps sont exécutés sur le noeud qui a la donnée
  - Sinon, notion de *rack* pour exprimer la proximité
- Ordonnancement
  - Hadoop ordonnance typiquement
    - 10-100 maps simultanés/noeud
    - 0.95 reduce/noeud
  - Configurable explicitement

Exemple

- WordCount
  - dénombrement des occurrences des mots
- Les k et v des <k, v> doivent être Writable pour être sérialisable par Hadoop
- Les clef (k) doivent être WritableComparable pour le tri lors du Shuffle

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

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

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Exemple – Map

- WordCount Map : génère une paire **<word, 1>** pour chaque mot

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

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Exemple – Reduce

- WordCount Reduce : génère un total **<word, n>** pour chaque mot

```
   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);
    }
  }
```

Exemple – Assemblage du Job

- Classes pour :
  - Input, Output : classes de sérialisation, chemin
  - Map et Reduce
  - Optionel : class Combiner
    - Reduce localement sur chaque noeud avant le Shuffle

```
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

Interface REST

- Monitoring de jobs
- Exemple :

```
curl --compressed -H "Accept:
application/json" -X GET
"http://host.domain.com:8088/proxy/applicatio
n_1326821518301_0010/ws/v1/mapreduce/jobs"
```


```
{
   "jobs" : {
      "job" : [
         {
            "runningReduceAttempts" : 1,
            "reduceProgress" : 72.104515,
            "failedReduceAttempts" : 0,
            "newMapAttempts" : 0,
            "mapsRunning" : 0,
            "state" : "RUNNING",
            "successfulReduceAttempts" : 0,
            "reducesRunning" : 1,
            "acls" : [
               {
                  "value" : " ",
                  "name" : "mapreduce.job.acl-modify-job"
               },
               {
                  "value" : " ",
                  "name" : "mapreduce.job.acl-view-job"
               }
            ],
            "reducesPending" : 0,
            "user" : "user1",
            "reducesTotal" : 1,
            "mapsCompleted" : 1,
            "startTime" : 1326860720902,
            "id" : "job_1326821518301_10_10",
            "successfulMapAttempts" : 1,
            "runningMapAttempts" : 0,
            "newReduceAttempts" : 0,
            "name" : "Sleep job",
            "mapsPending" : 0,
            "elapsedTime" : 64432,
            "reducesCompleted" : 0,
            "mapProgress" : 100,
            "diagnostics" : "",
            "failedMapAttempts" : 0,
            "killedReduceAttempts" : 0,
            "mapsTotal" : 1,
            "uberized" : false,
            "killedMapAttempts" : 0,
            "finishTime" : 0
         }
      ]
   }
}
```

Travail à faire

Exemple : WordCount

- Compilez et faites tourner l'exemple WordCount sur un corpus de texte
- Familiarisez-vous avec le code et le makefile
- Nous utiliserons le mode Single Node
  - Les plus téméraires peuvent essayer une configuration cluster sur plafrim
    - Mais il n'y aura pas de place pour tout le monde en même temps
  - Consultez la documentation Hadoop sur la configuration Cluster `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html`

Fouille de données diverses

- Fouille de données dans différents data sets
  - Repartez du code WordCount + makefile
- Vous ne ferez sans doute pas tous les exemples proposés :-)
  - Au choix

Analyse de code Java

- Adaptez le code WordCount pour l'appliquer à du code Java
  - Éliminez les délimiteurs et autres opérateurs
- Écrivez un code qui :
  - Énumère les identifiants
  - Construit un index des identifiants (identifiant -> fichiers)
  - Repère dans quels fichiers une classe est utilisée
    - Comment reconnait-on une classe sans parser ?

Musixmatch – comptage de mots (bis)

- Les fichiers mxm_dataset_* contiennent les occurences des mots les plus fréquents dans les textes de chansons
  - Données anonymisées, et pas de texte intégral (bien entendu)
- Quel est le mot le plus fréquent ?
- Quelle est la chanson la plus « commune » (moyenne de fréquence la plus élevée) ?
- Quelles sont les chansons qui utilisent le mot « *cilk* »

Statistiques de circulation

- Le fichier `Dodgers.data` contient les statistiques de circulation sur la bretelle de sortie « Glendale » de l'autoroute 101N à Los Angeles
  - Cumul du nombre de véhicules sur une tranche de 5 minutes
- Quelle tranche de 5 minute est la plus chargée ?
- Quelle heure / jour est le plus chargé ?
- Combien de véhicules sont passés le 24 juillet 2005 ?

Graphes d'adjacence

- Le répertoire `lkml/` contient les fils de réponse sur la mailing-list du noyau Linux
  - Données anonymisées, chaque personne est identifiée par un nombre
- Qui a posté le plus de messages ?
- Qui a reçu le moins de réponses ?
- Combien un message reçoit-il de réponses en moyenne ?

Analyse de données météo

- Télécharger quelques fichiers de la base météo américaine sur : `ftp://ftp.ncdc.noaa.gov/pub/data/asos-onemin/`
  - Uniquement des fichiers 6406-*, et pas tous !
- Format de fichier
  - Documentation `ftp://ftp.ncdc.noaa.gov/pub/data/asos-onemin/td6406.txt`
  - Résumé : entrées de 99 caractères/ligne
  - Exemple :

```
03856KHSV HSV2004090100000600 NP 0.00 39979 29.447 29.453 29.452 67 65
```

  - station 03856KHSV, date 20040901 (01/09/2004), précipitations 0.000 (inch), pression 29.452 (HgInch), température 67°F, point de rosée 65°F

Analyse de données météo

- Réaliser un code MapReduce qui extrait :
  - La date de la température la plus haute
  - La date de la température la plus haute de chaque année
  - Le nom de la station météo qui a reçu le plus de précipitation

À vous de jouer !

http://dept-info.labri.fr/~denis/