Jean-Daniel Cryans
Groupe d’intérêt Hadoop de l’ÉTS
2 juillet 2008

Mais... Qui suis-je?

- Finissant en génie logiciel.
- Membre du laboratoire d’architecture des systèmes informatiques (LASI).
- Récipiendaire d’une bourse de recherche de premier cycle du CRSNG sous la direction de Alain April et Roger Champagne.
- Contributeur actif de HBase, un sous projet de Hadoop.
- Fondateur du groupe d’intérêt Hadoop de l’ÉTS.

Objectifs des 2 présentations

- Présenter l'infrastructure de Google.
- Présenter l'alternative source libre.
- Sensibiliser les étudiants de l'ÉTS aux systèmes à l'échelle du web.
- Amener des étudiants à choisir Hadoop et ses projets pour leurs projets.

- Vous avez le droit de poser des questions tout au long de la présentation!

Hadoop et MapReduce : traitement distribué à l'échelle du web

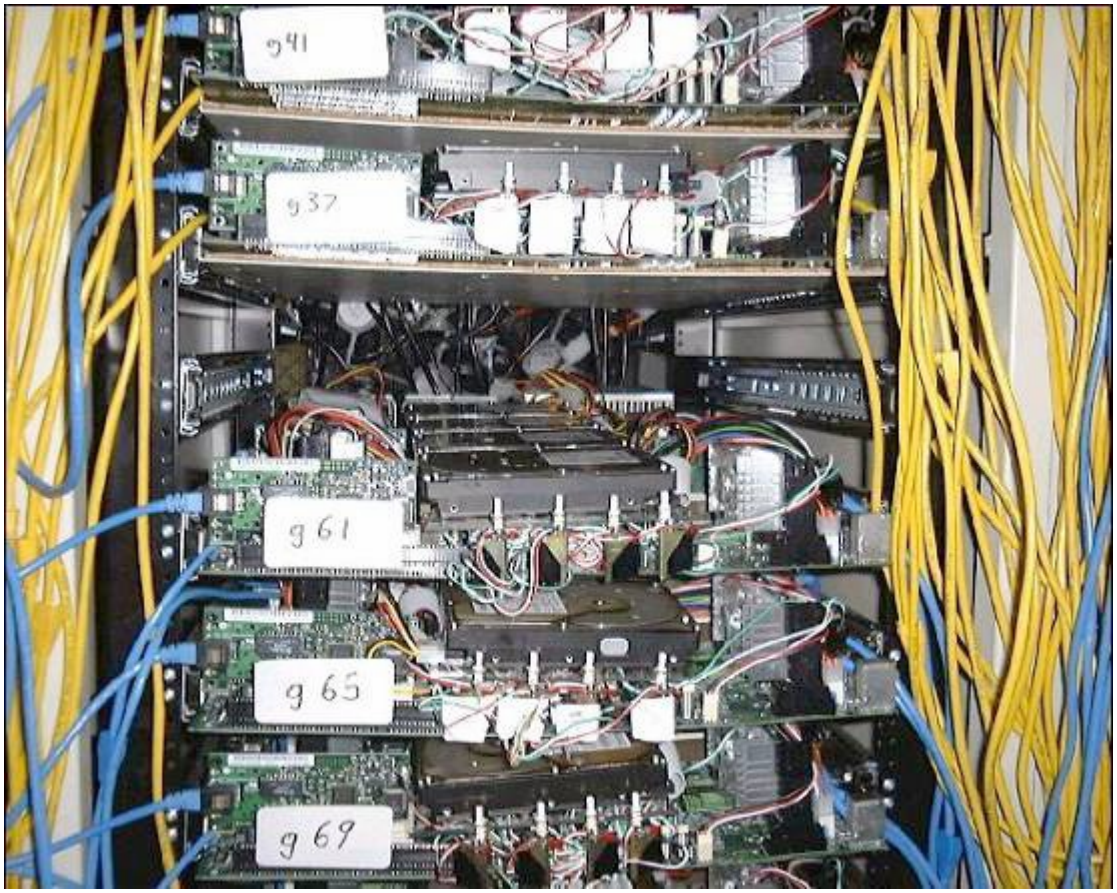
Références

- Hadoop : www.hadoop.com
- Hadoop Summit and Data-Intensive Computing Symposium : http://research.yahoo.com/node/2104
- Papiers de Google sur GFS et MapReduce : http://labs.google.com
- Highly-Available, Reliable and Scalable Systems: Exploring HBase and its Dependencies, papier soumit à HASE08 par Jean-Daniel Cryans, Alain April et Roger Champagne

Un peu d’histoire...

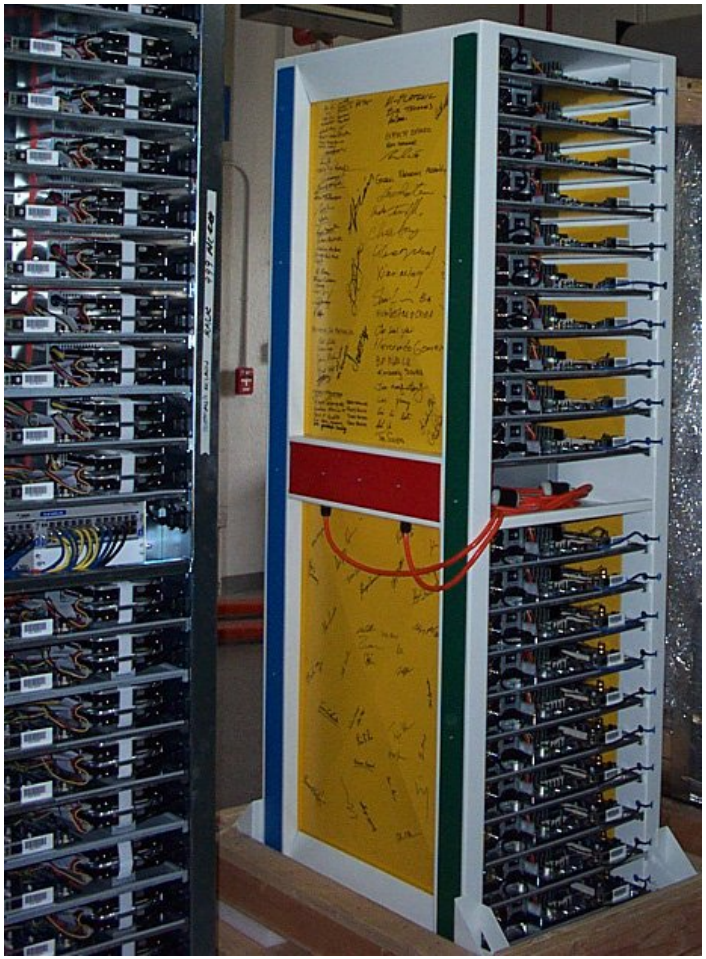
- 2002-2004
  - Nutch, un *web crawler* source libre est démarré par Doug Cutting. Ne pas parvient pas à indexer le web.
- 2004-2006
  - Google publie 2 papiers : le Google File System et MapReduce. Un système de fichier distribué est intégré à Nutch.
- 2006-2008
  - Yahoo! engage Doug Cutting et Hadoop devient un projet *top level* d’Apache et est maintenant propulsé par une équipe d’ingénieurs.
- 2008
  - Yahoo! met en production un cluster de plus de 10 000 processeurs pour son *Web Search*.

Pendant ce temps chez Google...



Le rack typique en 1999

Pendant ce temps chez Google...

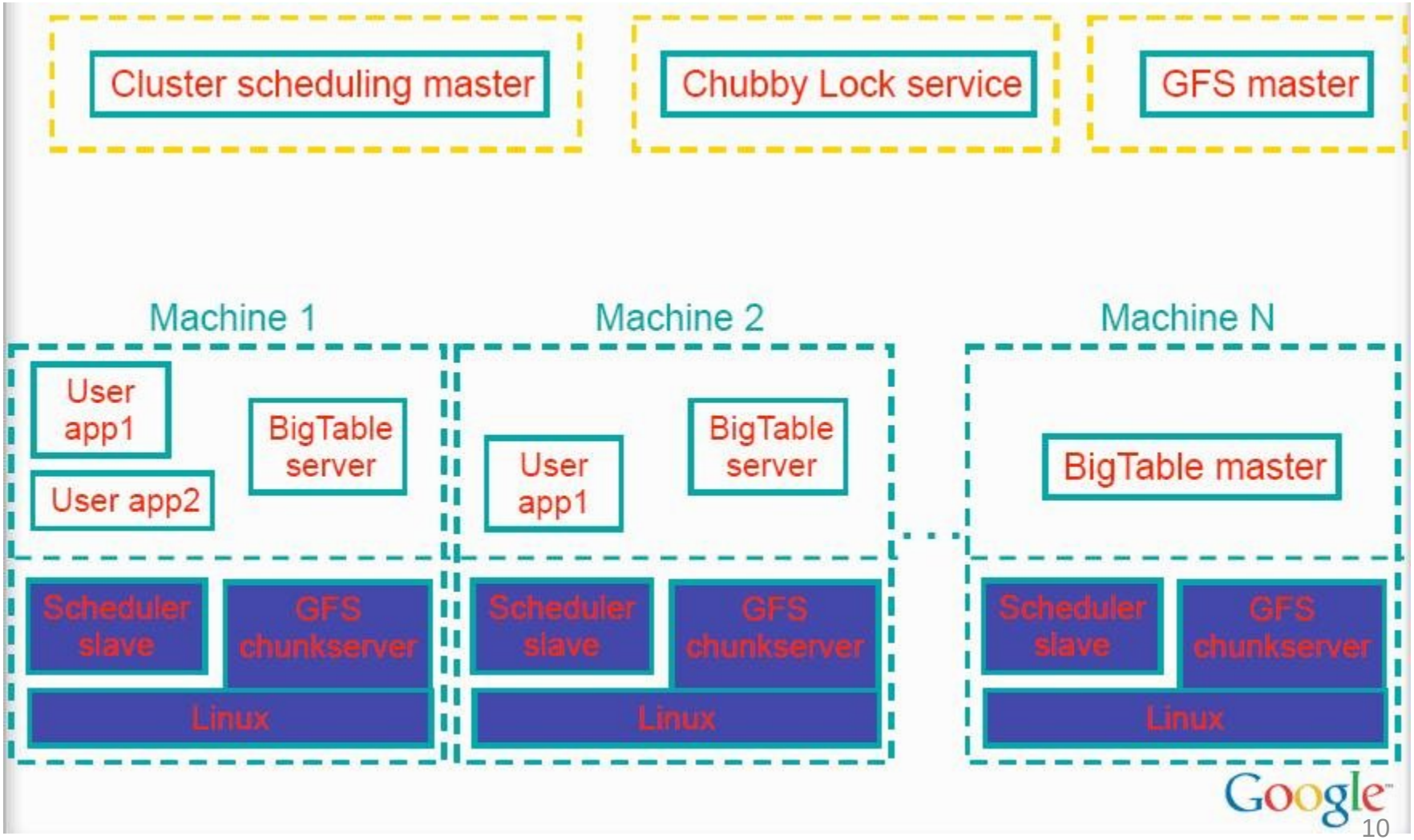


2008 : le modèle courrant, conçu en fonction des pannes.

Quelques définitions avant de commencer

- Google File System = Hadoop Distributed File System (HDFS)
- MapReduce, c’est un modèle de programmation
- Hadoop = HDFS + une implémentation de MapReduce
- 1 teraoctet = 1000 gigaoctet
- 1 petaoctet = 1000 teractet

Plan de match



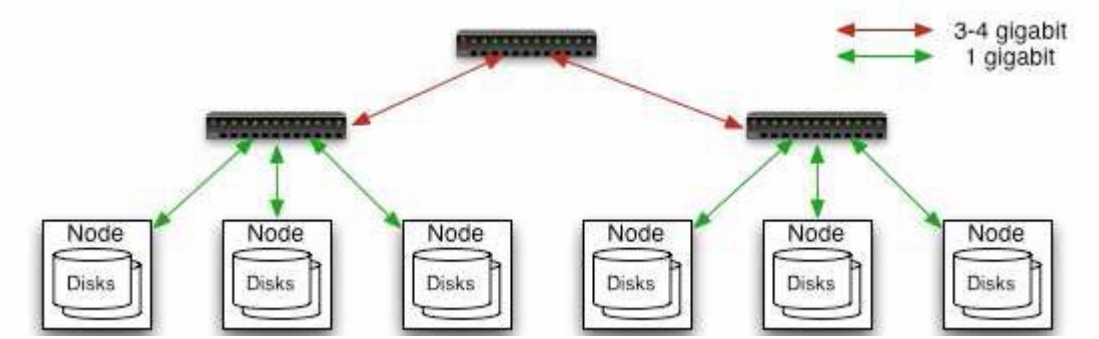
Pourquoi Hadoop?

- Besoin de traiter d'immenses ensembles de données sur d'immenses grappes de machines.
- Inclure la fiabilité dans les applications implique un coût élevé.
- Les noeuds ont des pannes quotidiennement
  - Les pannes sont normales, pas exceptionnelles.
  - Le nombre de noeuds dans une grappe n'est pas constant.
- Le besoin d'une infrastructure commune
  - Performante, fiable, facile à utiliser.
  - Source libre, license d'Apache.

Qui l’utilise?

- Amazon, AWS
- Facebook
- Google
- IBM
- Joost
- Last.fm
- New York Times
- Powerset
- Veoh
- Yahoo!
- Et tant d’autres...

Matériel de commodité



- Arbre à 2 niveaux
  - Les noeuds sont des PC!
  - 30-40 neuds par rack
  - Lien à 3-4GB à la sortie des noeuds
  - Lien à 1GB à l'intérieur des noeuds

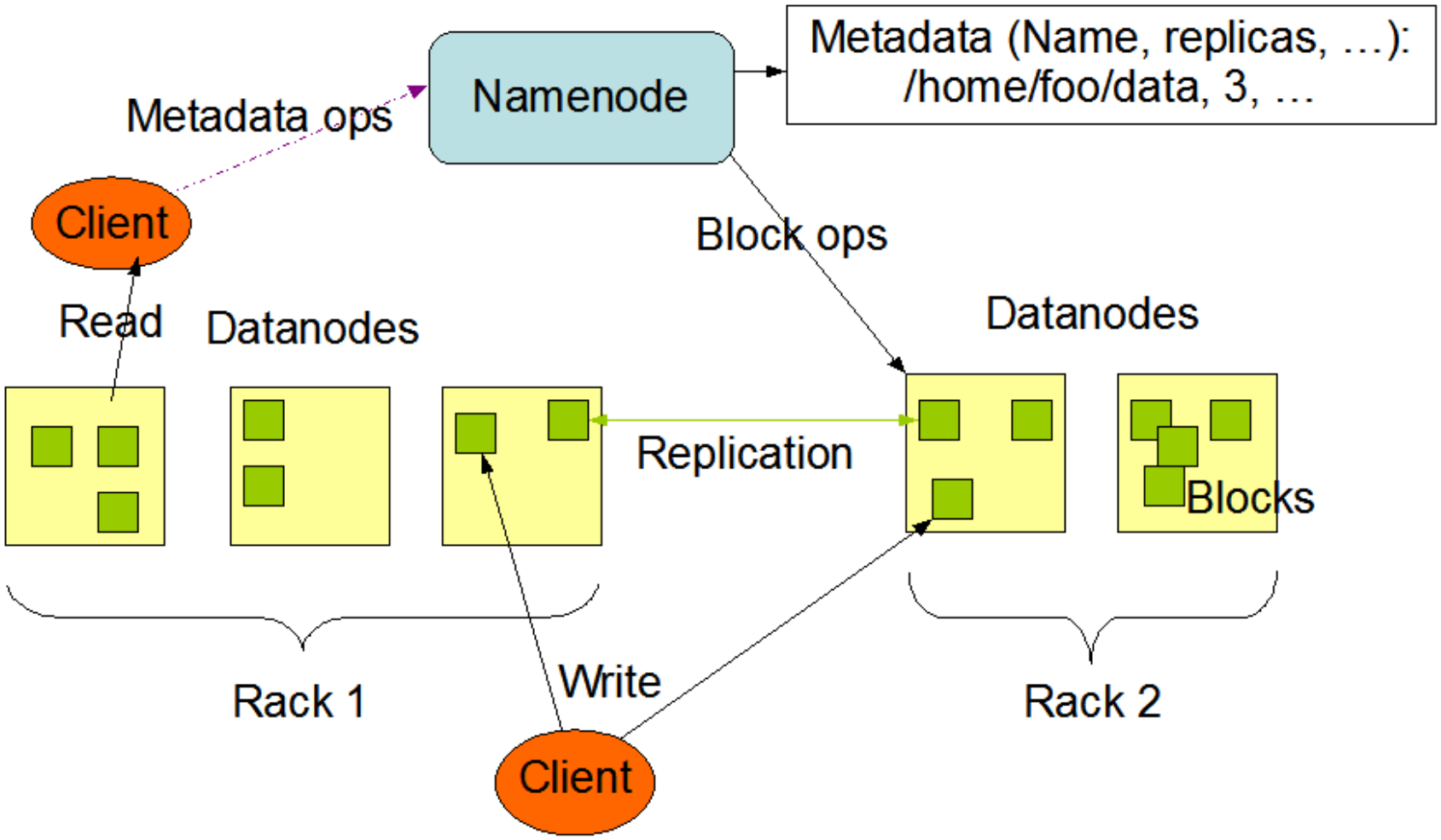
Les buts du HDFS

- Très grand système de fichier distribué.
  - 10 000 noeuds, 100 millions de fichiers, 10 PB.
- Tient compte du matériel de commodité.
  - Les fichiers sont répliqués.
  - Détecte les faute et en recouvre.
- Optimizé pour le traitement en lot.
  - Les calculs sont transférés où se situe les données.
  - Résultat : une bande passante agrégée très élevée.

Système de fichier distribué

- Un seul espace de nom (un disque virtuel).
- Cohérence des données
  - Modèle d'accès "écrire une fois, lire souvent".
  - Les clients peuvent seulement adjoindre aux fichiers existant.
- Fichiers séparés en blocks.
  - Par défaut, des blocks de 128MB.
  - Les blocks sont répliqués à travers les noeuds.
- Clients intelligents
  - Les clients peuvent trouver la location des blocks.
  - Les clients accèdent directement aux données.

HDFS Architecture



Les fonctions du Namenode

- Gère l'espace de nom
  - Mappe les fichiers à leurs blocks
  - Mappe les blocks aux Datanodes
- Gestion de la grappe de machines
- Engin de réplication des blocks

Les fonctions du Datanode

- Un serveur de blocks
  - Entreposer les données dans un système de fichier local comme par exemple *ext3*.
  - Entreproser les meta données des blocks comme le CRC.
  - Servir les blocks aux clients et au Namenode.
- Rapport des blocks
  - Périodiquement, il envoie un rapport de tous ses blocks au Namenode.
- Faciliter le *pipelining* des blocks
  - Transférer aux autres Datanode des données spécifiques notamment lors de la réplication.

Fiabilité

- Le Namenode envoie un *heartbeat* aux 3 secondes aux Datanode.
- Les blocks sont répliqués autant de fois que spécifié.
- Utilisation de CRC32.
- Validation des données obtenues, sinon rechercher un autre réplica.
- Utilisation de log pour tout.
- Panne du Namenode... C’est un point de défaillance unique! Ils vont intégrer Zookeeper.

Interface utilisateur

- Ligne de commande comparable à un *shell.*
- Site web local qui donne les informations vitales sur la santé de la grappe.
- Les fichiers peuvent être écris et obtenu directement par un URL.

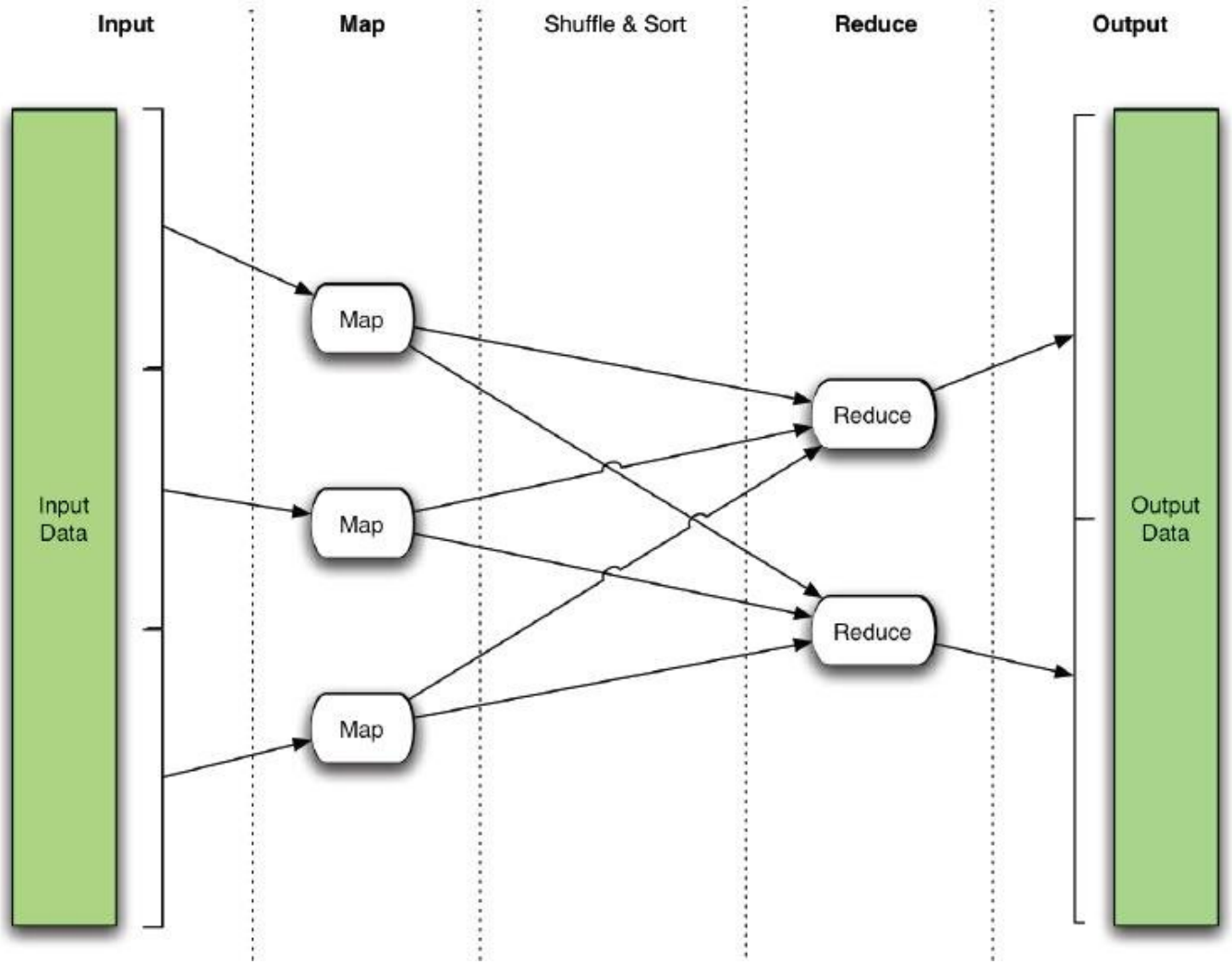
MapReduce

- Problème : lire 100 terabytes
- Prend 11 jours sur 1 seul ordinateur.
- Prend 15 minutes sur 1000 ordinateurs mais...
- Dans les grandes grappes, plusieurs ordinateurs ont des pannes!
- Comme pour HDFS, besoin d’avoir une infrastructure commune.

MapReduce

- MapReduce est un model efficace pour le traitement distribué.
- Ça fonctionne comme une pipeline Unix :
  - cat input | grep | sort | uniq -c | cat > output
  - **Input | Map | Shuffle & Sort | Reduce | Output**
- Efficace car :
  - Lecture en continu, réduit les *seeks.*
  - *Pipelining*
- S'applique partout :
  - Traitement de log
  - Indexage du Web
  - Traitement d'images
  - Apprentissage machine

Flux de données



Quelques fonctionnalités

- Les tâches Map et Reduce sont finement définies.
  - Améliore la balance des charges.
  - Recouvrement plus rapide lors des pannes.
- Ré-exécution automatique
  - Dans une grand grappe, il y a toujours des machines plus lentes.
  - Hadoop relance les tâches qui ont échoué ou qui prennent trop de temps.
- Optimisation de la localisation
  - Avec autant de données, on peut facilement saturer la bande passante.
  - Avec MapReduce, les tâches Map sont exécutées sur les machines où se trouvent les données tant que c'est possible.

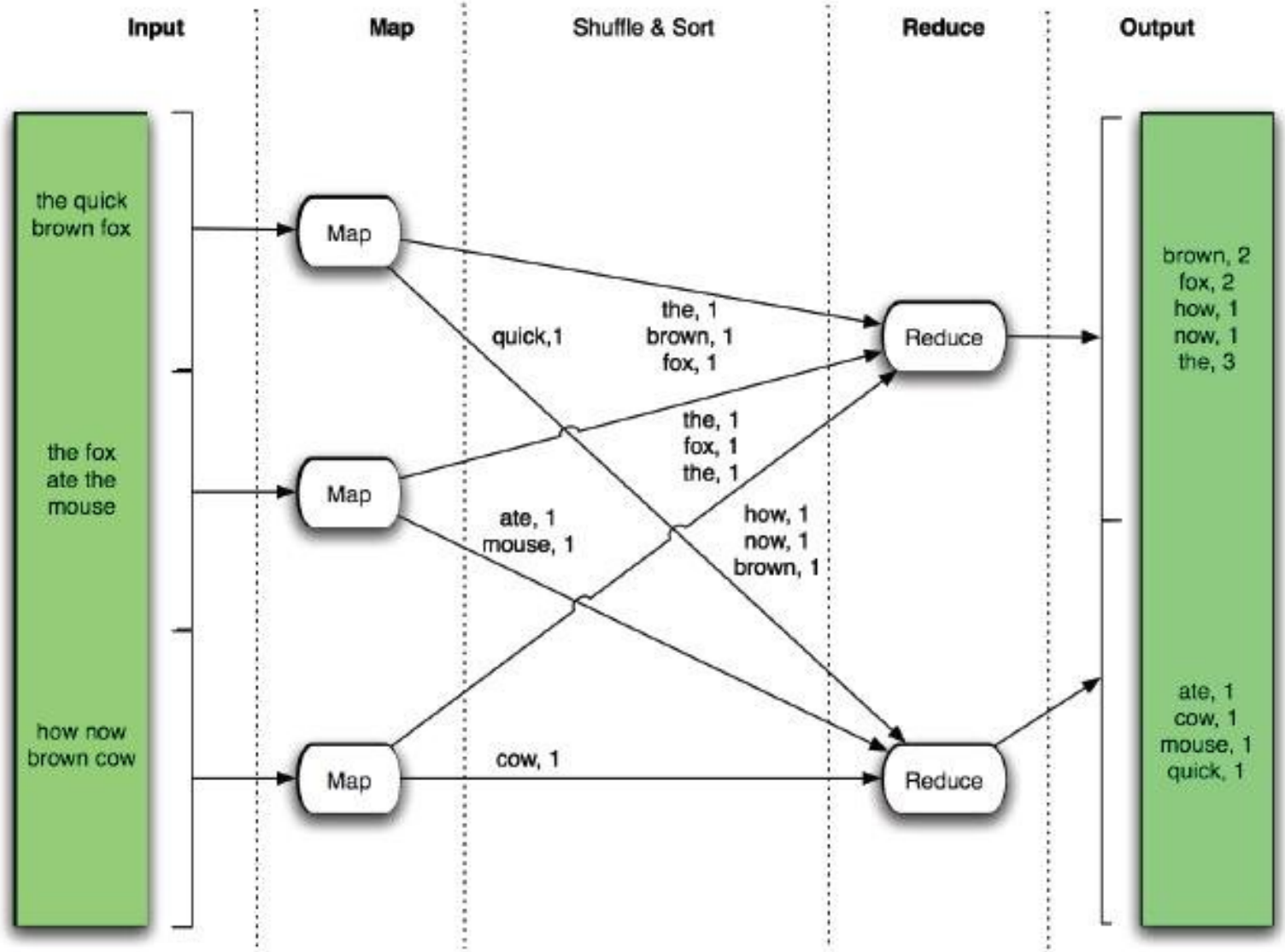
Map et Reduce

- Mapper :
  - Entrée : une portion de données dans un format prédéfini.
  - Sortie : une clé et une valeur.
- Reducer :
  - Entrée : une clé et un ensemble de valeurs.
  - Sortie : une clé et une valeur.
- Pour lancer le programme :
  - Définir la job
  - Lancer la job dans la grappe!

Exemple : compteur de mot

- Mapper :
  - Entrée : lignes de texte
  - Sortie : clé : mot, valeur : 1
- Reducer :
  - Entrée : clé : mot, ensemble de compte
  - Sortie : clé : mot, valeur : somme des comptes

Flux de données pour l'exemple



Le code du Mapper

```
public static class MapClass extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, IntWritable> {

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

  public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
```

Le code du Reducer

```
public static class Reduce extends MapReduceBase
      implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
          OutputCollector<Text, IntWritable> output,
    Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Et pour lancer la tâche

```
public class WordCount {
……
  public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    // the keys are words (strings)
    conf.setOutputKeyClass(Text.class);
    // the values are counts (ints)
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(MapClass.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputPath(new Path(args[0]);
    conf.setOutputPath(new Path(args[1]);
    JobClient.runJob(conf);
```

Dans les autres langages

- Le *Streaming* permet de passer du bash, du Python ou même du Perl. C'est plus lent par contre…
- Pour le C++, il existe une librairie spéciale qui permet d'avoir les mêmes performances.
- Yahoo! a inventé un langage appelé Pig qui permet de presque faire du SQL, donc des tâches plus haut niveau.

Exemple de code avec Pig

```
input = LOAD ’in-dir' USING TextLoader();
words = FOREACH input GENERATE
    FLATTEN(TOKENIZE(*));
grouped = GROUP words BY $0;
counts = FOREACH grouped GENERATE
   group,
    COUNT(words);
STORE counts INTO ‘out-dir’;
```

MapReduce chez Yahoo!

- Ils ont environ 10 000 machines dédiées.
- Leur plus gros a 2 000 noeuds.
- Ils y entreposent plus de 1 petaoctet.
- Ils passent environ 10 000 jobs par se

Dans le prochain épisode

- HBase, une base de données distribuée qui utilise Hadoop.
- Basé sur Bigtable, le logiciel qui permet à Google de faire fonctionner Google Analytics, Google Maps, YouTube, etc.
- Présentation du projet de recherche mené cet été par Jean-Daniel Cryans.

Des questions?