Map-Reduce: pour quoi ? pour qui ?

F. Suter

Centre de Calcul de l'IN2P3

Journée Cloud – 30 novembre 2012

Avant de parler Map-Reduce . . .

- Le Big Data : pour quoi ? pour qui ?

Avant de parler Map-Reduce . . .

- Le Big Data : pour quoi ? pour qui ?
  - Autant de définitions/solutions que de domaines

Avant de parler Map-Reduce . . .

- Le Big Data : pour quoi ? pour qui ?
  - Autant de définitions/solutions que de domaines
- Big Data en HEP
  - Beaucoup de fichiers indépendants (ou presque)
  - HTC : 1 job = 1 core
  - Solution : Grille de calcul/données

Avant de parler Map-Reduce . . .

- Le Big Data : pour quoi ? pour qui ?
  - Autant de définitions/solutions que de domaines
- Big Data en HEP
  - Beaucoup de fichiers indépendants (ou presque)
  - HTC : 1 job = 1 core
  - Solution : Grille de calcul/données
- Big Data en HPC
  - Gros jobs parallèles : milliers de cœurs ⇒ grosse mémoire distribuée
  - Big Data ⇒ gros traitements en *in-core*
  - Solution : I/O haute performance (disque et réseau)

Avant de parler Map-Reduce . . .

- Le Big Data : pour quoi ? pour qui ?
  - Autant de définitions/solutions que de domaines
- Big Data en HEP
  - Beaucoup de fichiers indépendants (ou presque)
  - HTC : 1 job = 1 core
  - Solution : Grille de calcul/données
- Big Data en HPC
  - Gros jobs parallèles : milliers de cœurs ⇒ grosse mémoire distribuée
  - Big Data ⇒ gros traitements en *in-core*
  - Solution : I/O haute performance (disque et réseau)
- Big Data Analystics (implicite pour beaucoup de monde)
  - indexation du web, fouille de données, analyse de logs, machine learning, analyse financière, . . .
  - Solution : Map-Reduce

Map-Reduce : c'est quoi ?

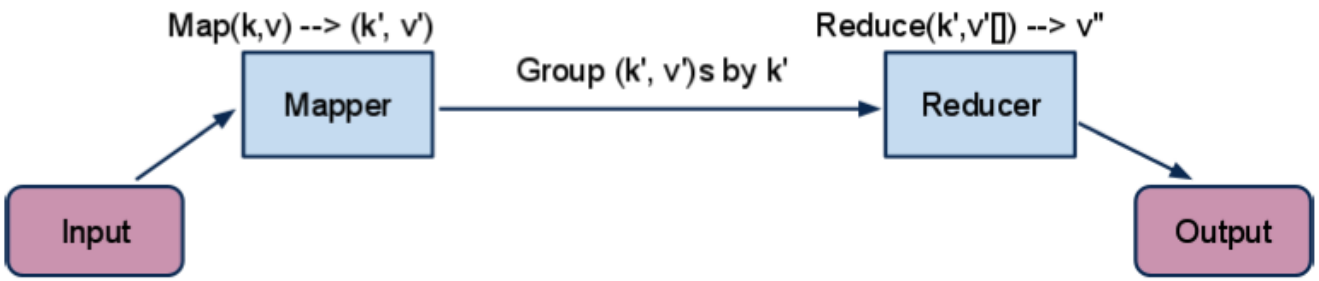
- Un modèle de programmation pour les applications data-intensives
  - Proposé par Google dès 2004
  - Simple, inspiré par la programmation fonctionelle
  - Brique de base pour d'autres langages de programmation parallèle
  - Très extensible
  - Conçu pour les grands clusters
    - Ressources sujettes aux pannes et non fiables

J. Dean And S. Ghemawat. *MapReduce : Simplified Data Processing on Large Clusters*. In Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI). pp. 137–150. San Francisco, CA, december 2004.

Map-Reduce en Lisp

- `(map f (list l1 ...l2))`
  - `f` est un opérateur unaire
  - Exemple : `(map square (1 2 3 4))`
  - Résultat : `(1 4 9 16)`
- `(+ 1 4 9 16)`
  - `+` est un opérateur binaire
  - Décomposition : `(+ 1 (+ 4 (+ 9 16)))`
  - Résultat : 30

Map-Reduce à la Google



- `Map(key,val)` exécuté sur chaque item dans l'ensemble de données
  - Émet de nouvelles paires `(key, val)`
- `Reduce(key,val)` exécuté sur chaque clé unique générée par les map
  - Produit le résultat final

Un exemple : WordCount

- En entrée : un grand nombre de documents textuels
  - Des pages web, par exemple
- En sortie : le nombre d'occurrences de chaque mot
- Principe
  - Mapper : pour chaque mot `m`, renvoyer `(m, 1)`
  - Reducer : additionner les occurrences de `m` et renvoyer `(m, count)`

Mapper

```
map (String key, String value):
  for each word w in value:
    EmitIntermediate (w, "1");
```

Reducer

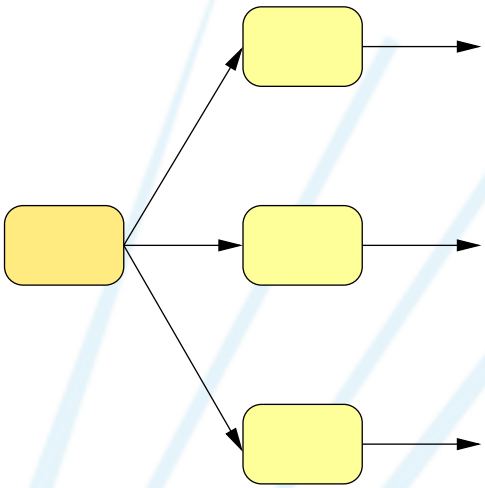
```
reduce (String key, Iterator values):
  int word_count = 0;
  for each v in values:
     word_count += ParseInt(v);
  Emit(key, AsString(word_count));
```

De façon plus générale

- Lire beaucoup de données
- Map : extraire un point intérêt de chaque entrée
- Shuffle and Sort
  - Équilibrage de charge
- Reduce : agréger, sommer, filtrer, ou transformer
- Écrire les résultats

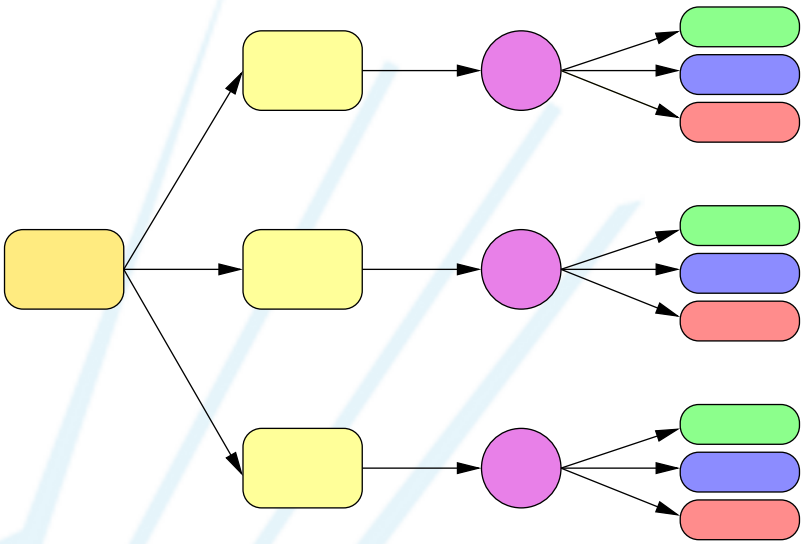
Exécution parallèle

- Distribute



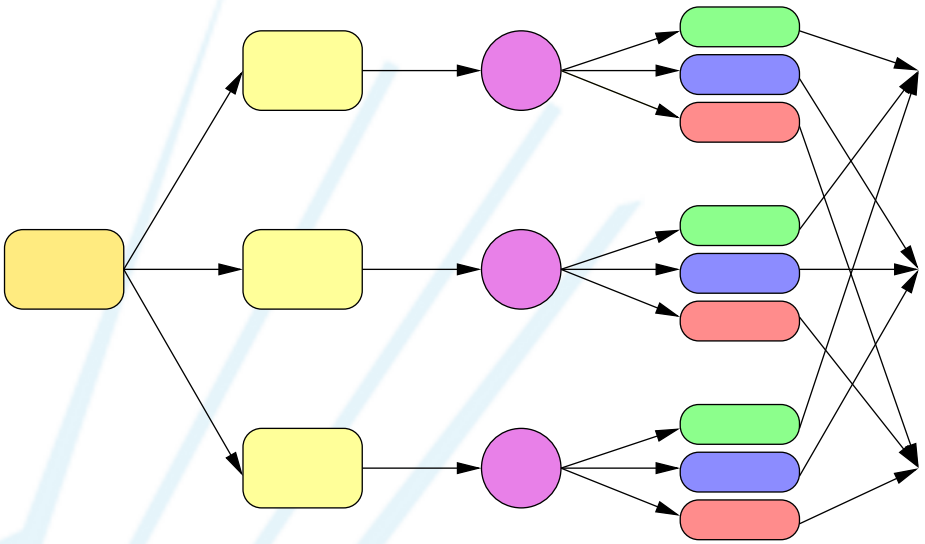
Exécution parallèle

- Map



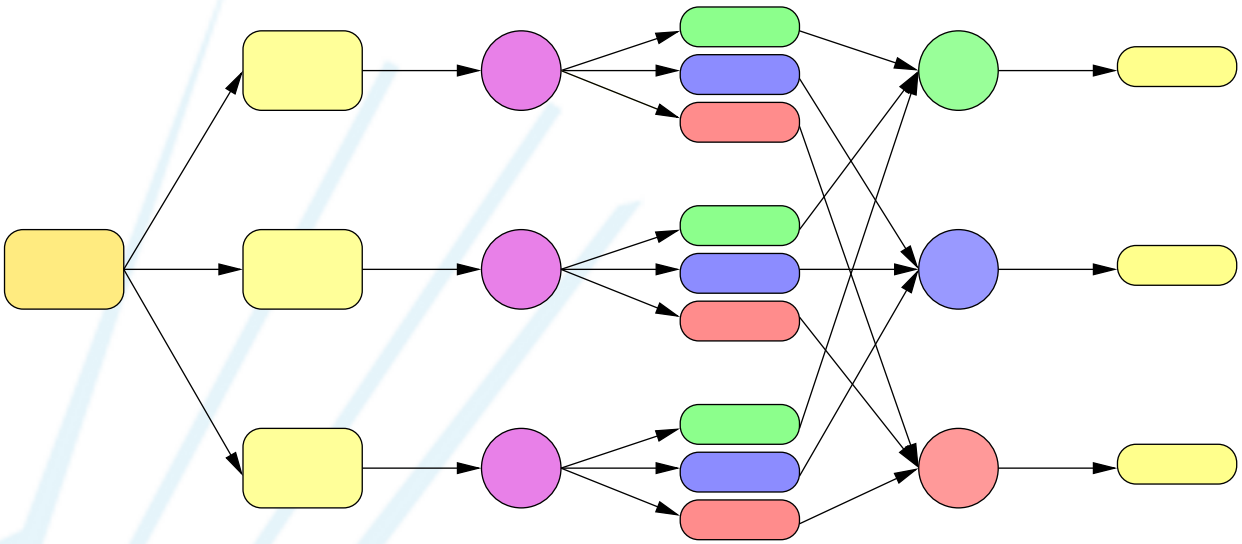
Exécution parallèle

- Shuffle and sort



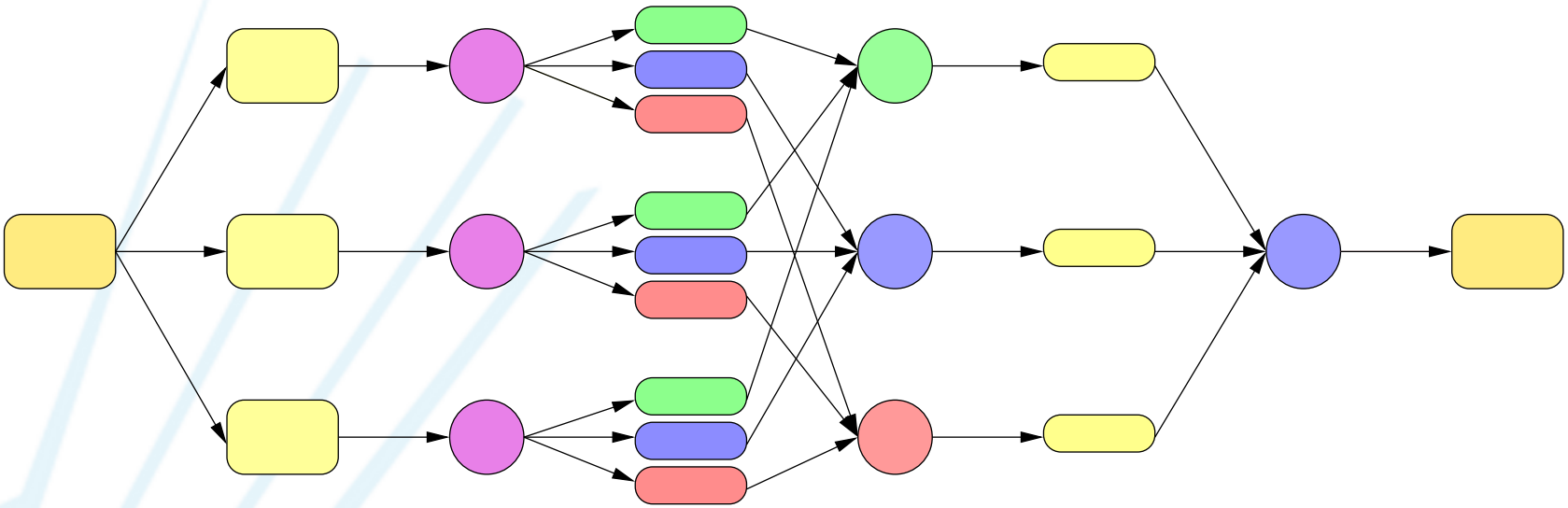
Exécution parallèle

- Reduce



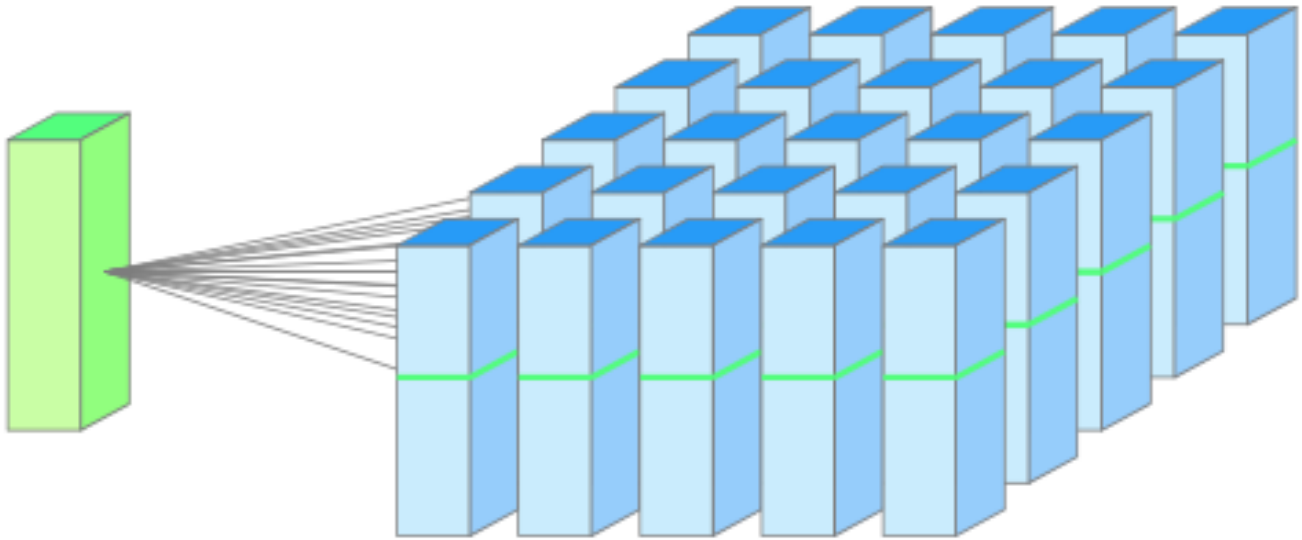
Exécution parallèle

- Combine



Accès aux données

- Performances ⇔ Système de fichiers distribué
  - GFS (Google File System), HDFS (Hadoop Distributed File System)



Runtime Map-Reduce

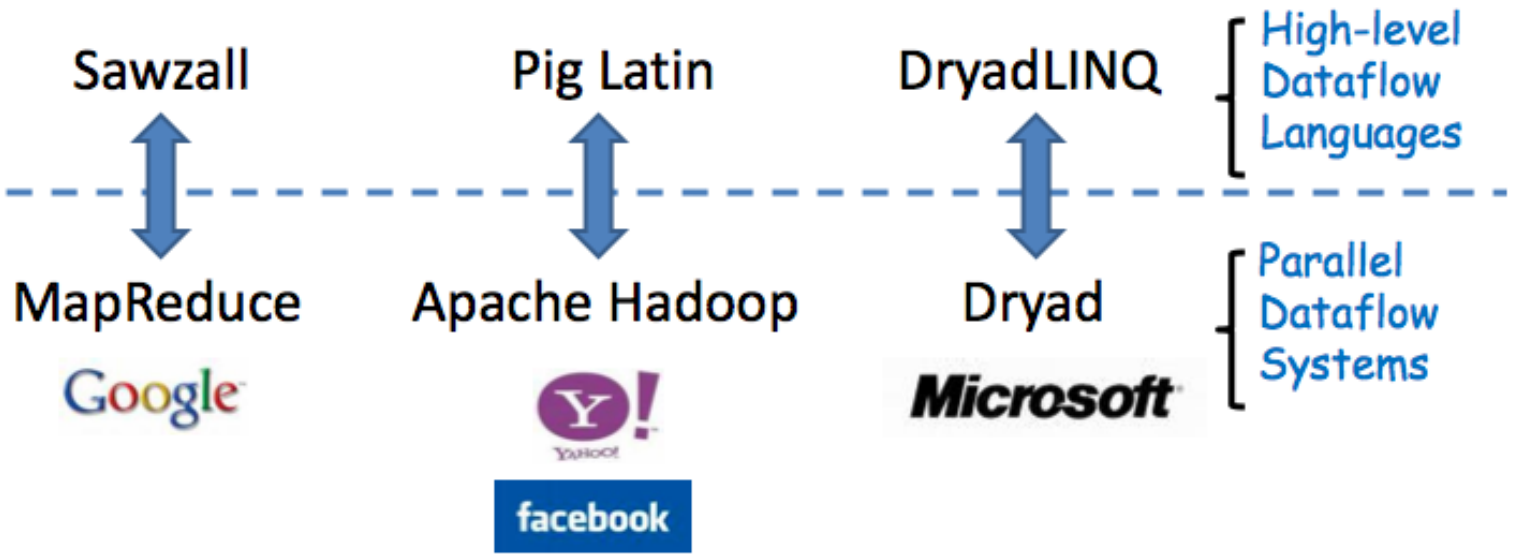
- Un runtime Map-Reduce gère de façon transparente
  - La parallelisation
  - L'équilibrage de charhe
  - Les I/O (réseau et disque)
  - Les pannes de machines
  - Les retardataires
- Des implémentations
  - Google MapReduce
    - Propriétaire, Code C++ (+ bindings Java et Python) fermé
    - Lié fortement à GFS
  - Hadoop
    - Open Source (Apache)
    - Cascading, Java, Ruby, Perl, Python, PHP, R, ou C++
    - Utilisé par Yahoo !, Facebook, Amazon et IBM
  - Et bien d'autres
    - Dryad (M$), Phoenix, Twister, Mars (GPU), FPMR (FPGA), ...

Au delà du WordCount : WordFrequency

- En entrée : un grand nombre de documents textuels
- En sortie : la fréquence d'occurrences de chaque mot
- Version naïve
  - Utiliser red Map-Reduce
  - MR1 : Compter les occurrences de tous les mots
    - Utilise des combiners
  - MR2 : Compter les occurrences de chaque mot et diviser par le total issu de MR1
- Optimisations
  - Utiliser la garantie d'ordre de traitement des clés reduce
  - Utiliser la fonction `EmitToAllReducers`
  - Principe
    - Chaque Map calcule son total de mots et l'associe à la clé ""
    - Envoie cette paire à tous les reducers
    - qui traitent cette clé en premier ⇒ nombre total de mots

Encore au delà : langages d'analyse

- Map-Reduce comme une brique de base



Exemple : PIG

SQL

```
SELECT category, AVG (pagerank)
FROM urls
WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10E6
```

PIG

```
Good_urls = FILTER urls BY pagerank > 0.2;
Groups = GROUP good_urls BY category;
Big_groups = FILTER groups BY COUNT(good_urls) > 10E6;
Output = FOREACH big_groups GENERATE category, AVG(good_urls.pagerank)
```

Conclusion

- Map-Reduce est un modèle de programmation à la mode
  - Bien adapté aux applications data-intensives
  - Facilement parallélisable
- C'est une des réponses à une des incarnations du Big Data
  - indexation du web, fouille de données, analyse de logs, machine learning, analyse financière, ...
- Pas la solution à tout non plus
- Domaine de recherche très actif