Exécution Map Reduce et Algèbre Spark

Master DAC – Bases de Données Large Echelle
Mohamed-Amine Baazizi
baazizi@ia.lip6.fr
2019-2020

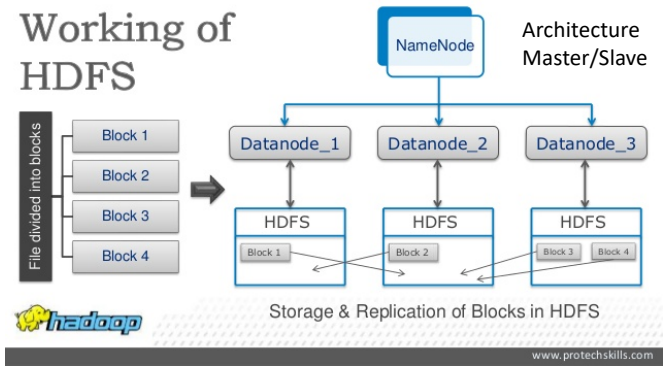
Plan

- Aperçu Hadoop file system et Map Reduce
- Exécution algèbre RDD dans Spark

HDFS (HaDoop File System)

- Système de gestion de données distribuées
- Passage à l'échelle (Peta octets, 4500 nœuds)
- Tolérance aux pannes grâce à la réplication
- Optimisé pour les lectures, écritures rares
- Fichier = plusieurs blocks
  - taille standard 128 MB
  - facteur de réplication 3, distribution sur différents nœuds

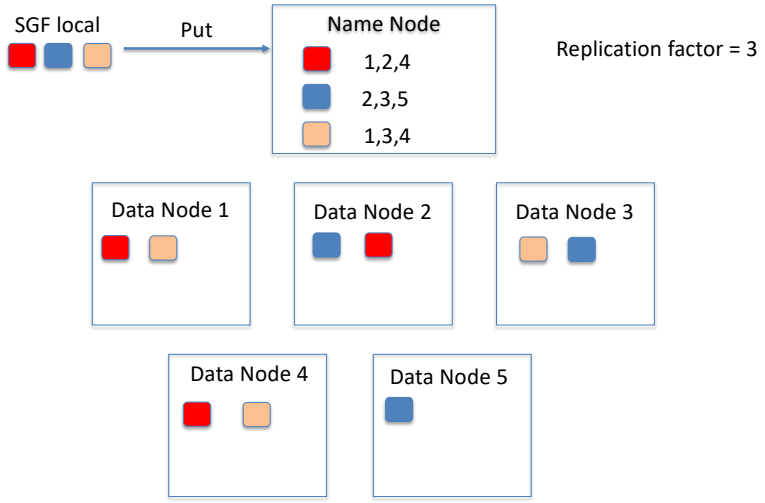
Architecture HDFS



Architecture HDFS : composants

- Un NameNode par cluster :
  - Contient métadonnées pour localiser les blocs
- Un DataNode par nœud
  - création, suppression, réplication, lecture et écriture de blocks sous l'ordre du NameNode
- Etapes de création d'un fichier
  - consulter NameNode pour disponibilité
  - découpage en blocs et envoi aux DataNode
  - demande de réplication

Illustration de HDFS



Démo HDFS

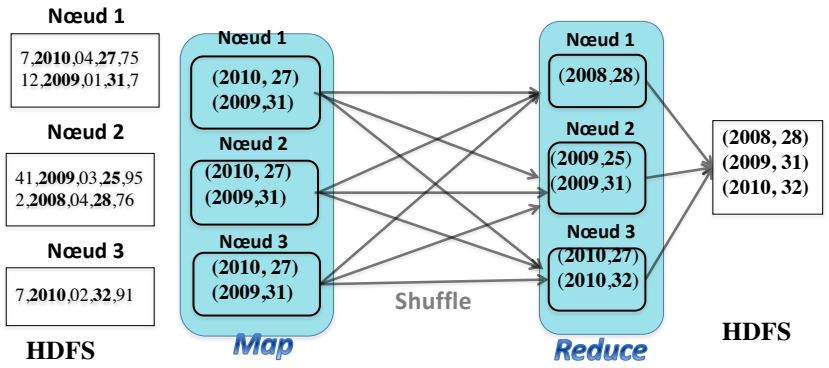
```
home$ hadoop fs -ls –h /tpch/lineitem.tbl
-rw-r--r--   3 bdle supergroup    718.9 M ...

home$ hdfs fsck /tpch/lineitem.tbl

Total size:753849433 B
 Total dirs:       0
 Total files:      1
 Total symlinks:             0
 Total blocks (validated): 6 (avg. block size 125641572 B)
 Minimally replicated blocks:  6 (100.0 %)
 Over-replicated blocks:  0 (0.0 %)
 …
 Default replication factor:     3
 ..
 Number of data-nodes:         5
 Number of racks:          1
```


Démo HDFS

```
home$ hdfs fsck /tpch/lineitem.tbl  -blocks -locations -files

0. BP-Number-IPAddr-Number: len=134217728 repl=3 [Datanode1,
Datanode2, Datanode2]

1. BP-Number-IPAddr-Number: len=134217728 repl=3 [Datanode1,
Datanode2, Datanode2]

…

5. BP-Number-IPAddr-Number: len=134217728 repl=3 [Datanode1,
Datanode2, Datanode2]
```

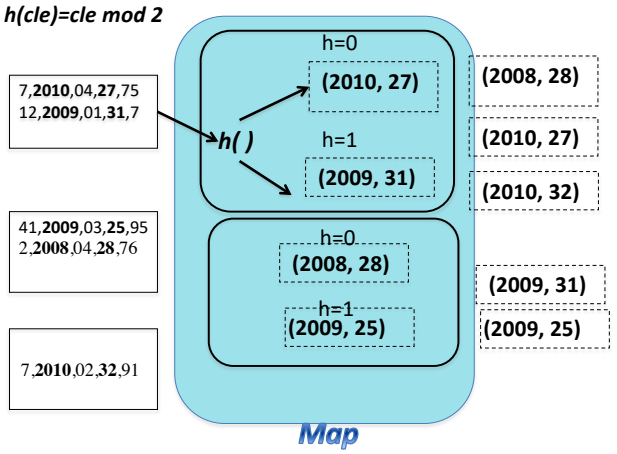
Possibilité d'utiliser interface graphique

Exécution **Hadoop** Map Reduce

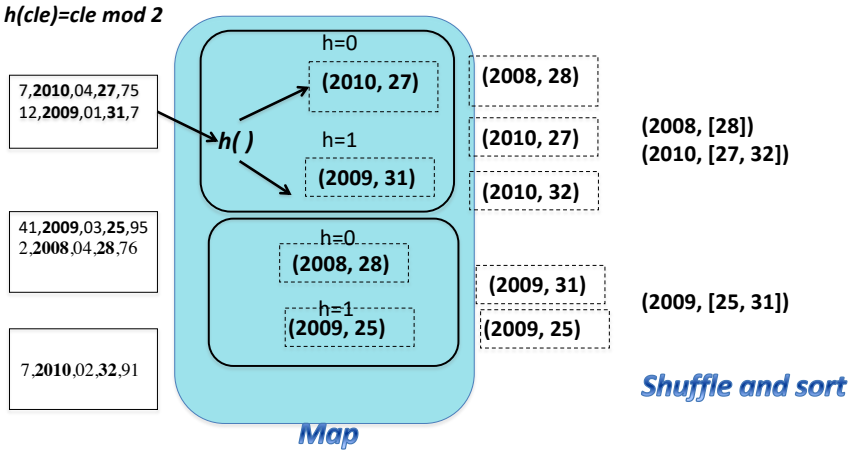


Entrée : n-uplets (station, **annee**, mois, **temp**, dept)
Résultat : select annee, Max(temp) group by annee

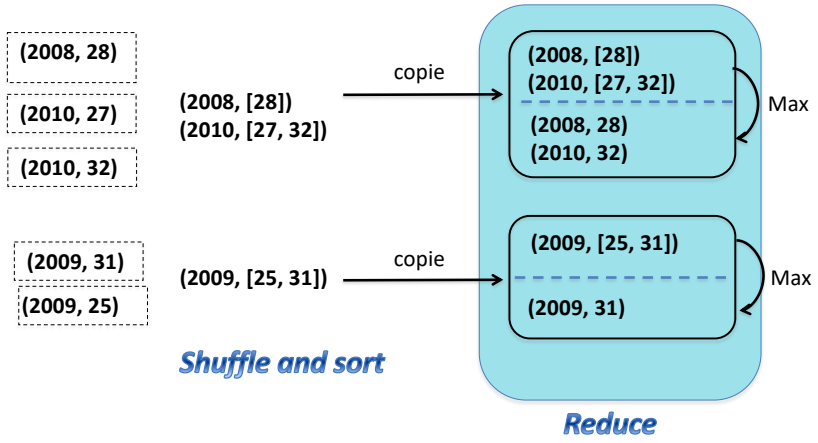
Phase Map



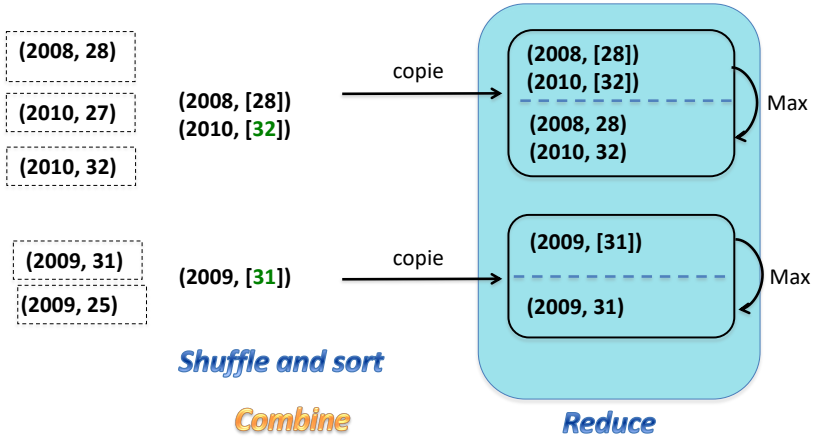
Phase Shuffle & Sort



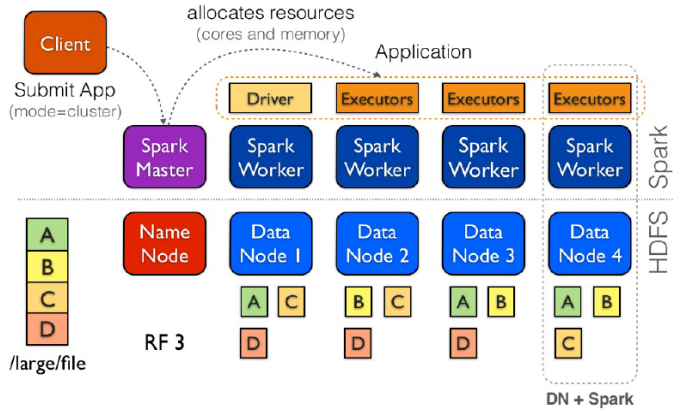
Phase Reduce



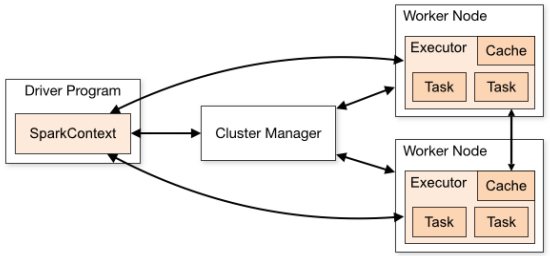
Combine



Spark avec HDFS

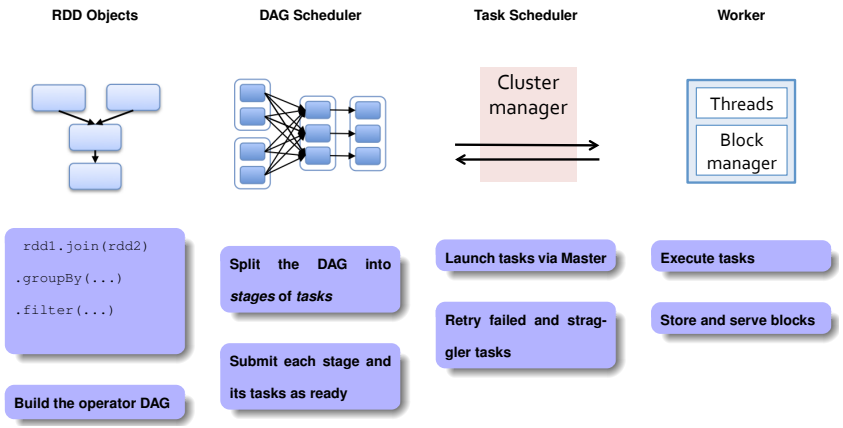


Composants Spark



- Driver : prog. utilsant API Spark pour spécifier les calculs d'une application
- Executor : processus lancé par une application, un par worker (par défaut)
- Task : unité d'exécution réalisée par un executor

Cycle de vie d'un programme

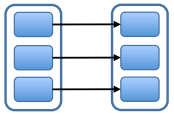


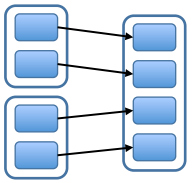
Terminologie

- Opération
  - Transformation : crée une nouvelle RDD à partir d'autre(s) RDD
    - retourne RDD du type de la transformation (MappedRDD, …)
    - locale (map, filter) ou distribuée (join, reduceByKey)
  - Action : évalue la RDD en exécutant la chaine de transformation
    - retourne type de base ou *User Defined Type*
- Stage : Séquence de transformations locales terminée par une transformation distribuée ou par une action
  - exécution *pipelined* des transformations locales
- Plan : Séquence de *stages* terminée par une action
- Pendant shuffle matérialisation de données intermédiaires

Transformations locales

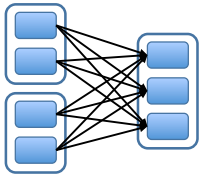
- Application sur les partitions locales
  - pas de shuffle
  - Ex.: map, filter, union, flatMap, mapValues

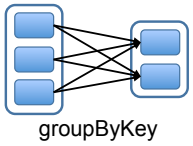




Transformations distribuées

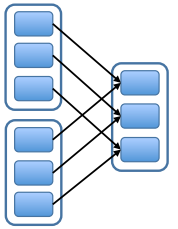
- Accès requis à toutes les partitions
  - Suffle requis
  - join, reduceByKey, groupByKey, distinct, intersect



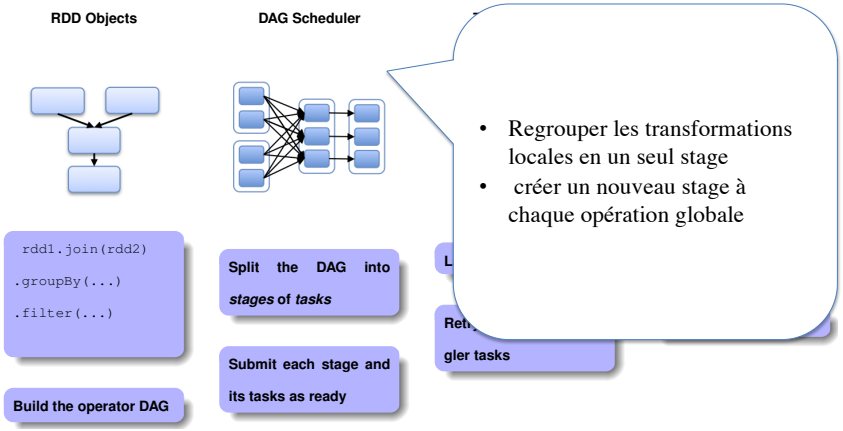


Optimisation

- Tirer profit du partitionnement effectué par transformation antérieure
  - jointure sur une clé pour laquelle les deux relations sont déjà partitionnées

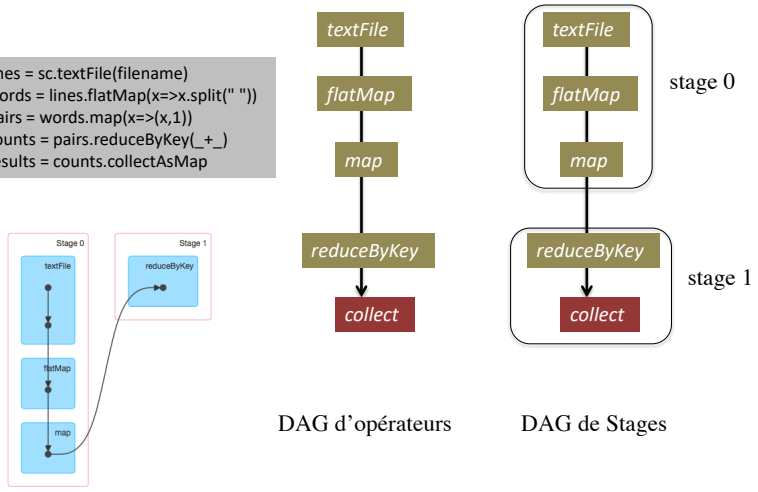


Génération du plan d'exécution

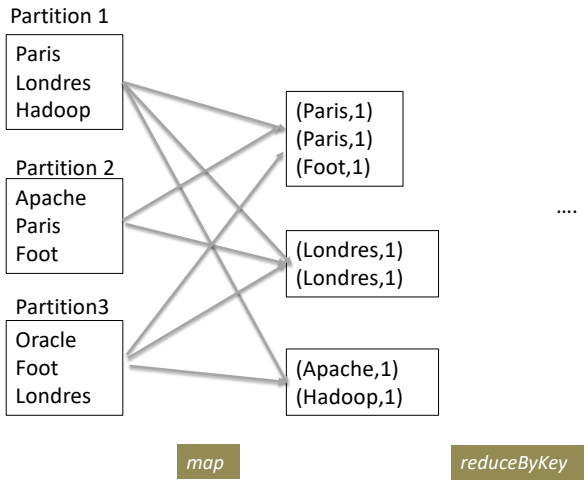


Wordcount

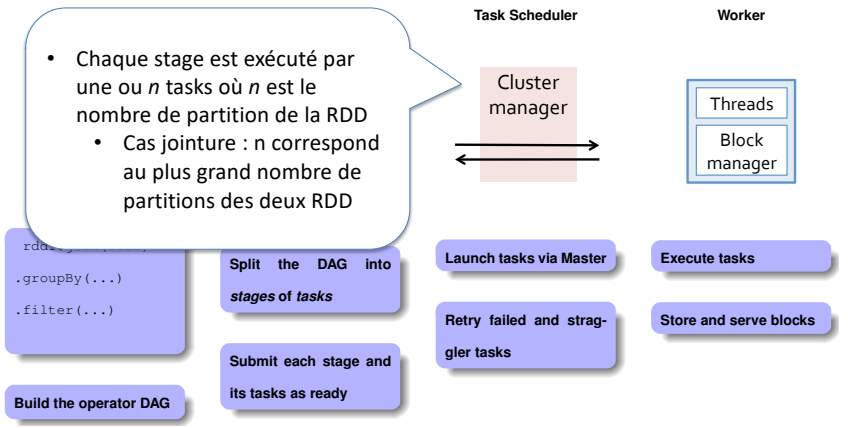
```
val lines = sc.textFile(filename)
val words = lines.flatMap(x=>x.split(" "))
val pairs = words.map(x=>(x,1))
val counts = pairs.reduceByKey(_+_)
val results = counts.collectAsMap
```



Wordcount illustré



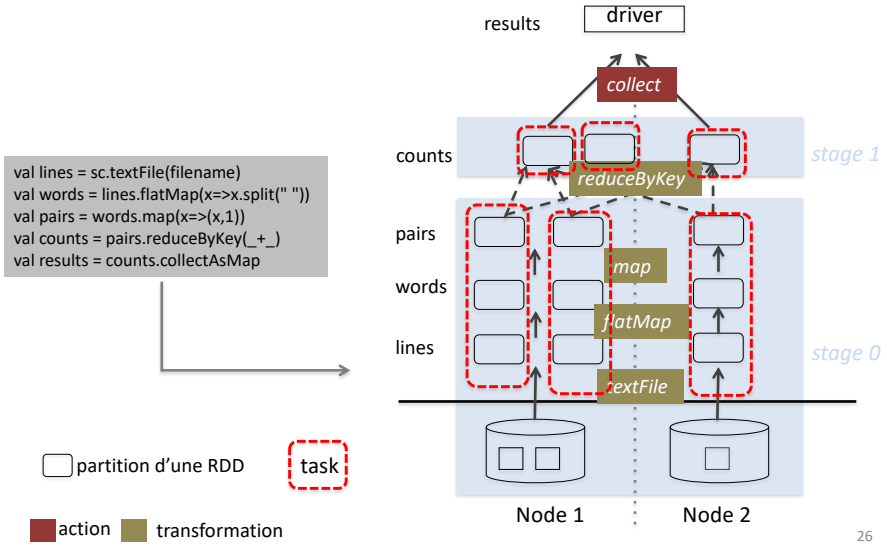
Exécution du plan



Exécution du plan

- Affectation de *tasks* aux stages en privilégiant la localité des données
  - exécuter une tâche dans le nœud de la partition
    - si partition dans HDFS, alors tâche du même dataNode
- Matérialisation des résultats produits pour chaque stage
  - en cas de panne, par exemple perte de certaines partitions, ne recalculer que celles-ci

Exécution du plan Wordcount



Scénario réel

config. matériel : 5 nœuds de calcul+stockage, 20 cœurs/nœud
Données tiennent sur 22 blocs HDFS

Noeud	Nb. tasks	Input (Mo)	Shuffle Write (Mo)	Temps
1	5	640.3	41.3	1.2 min
2	4	512.3	32.8	49 s
3	4	494.2	32.8	47 s
4	4	512.3	34.3	46 s
5	5	640.3	41.9	1.2 min
Total	*22*	*~2.799*	*181.1*	

Stage 0
ShuffleMapStage

Noeud	Nb. tasks	Shuffle Read (Mo)	Temps
1	11	91.6	1.8 min
4	11	91.5	1 min
Total	*22*	*181.1*	

Stage 1
ResultStage

Taille Shuffle write réduite
car utilisation systématique du *combiner*

Optimisation manuelle

- Stage 1 utilise 22 tasks sur petites partitions
- fusionner les partitions pour réduire le nombre de tasks

coalesce(numPartitions: Int, shuffle: Boolean = false, partitionner …)
quand *shuffle=true* possibilité de déplacer les données

```
val lines = sc.textFile(filename)
val words = lines.flatMap(x=>x.split(" "))
val pairs = words.map(x=>(x,1))
val counts =
pairs.reduceByKey(_+_).coalesce(2,
false)
val results = counts.collectAsMap
```

Noeud	Nb. tasks	Shuffle Read (Mo)	Temps
1	1	91.6	23 s
4	1	91.5	25 s
Total	*2*	*181.1*	

Stage 1

Bilan Algèbre RDD

- Algèbre riche
- Optimisation limitée à la notion de stages
  - Pas de notion de plan logique
  - Code utilisateur difficile à optimiser contrairement aux langages déclaratifs comme SQL
  - Absence de modèle de coût
  - Faible performances des agrégations

Présentation Devoir Maison

http://www-bd.lip6.fr/wiki/site/enseignement/master/bdle/start