BDLE

Exécution parallèle et distribuée

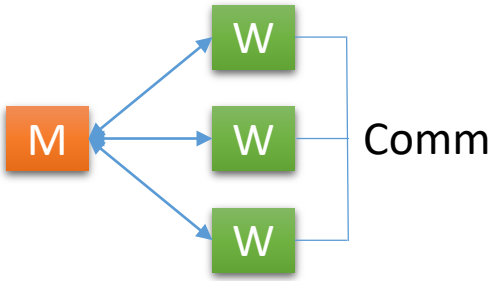
2020

Architecture et exécution

dans Spark

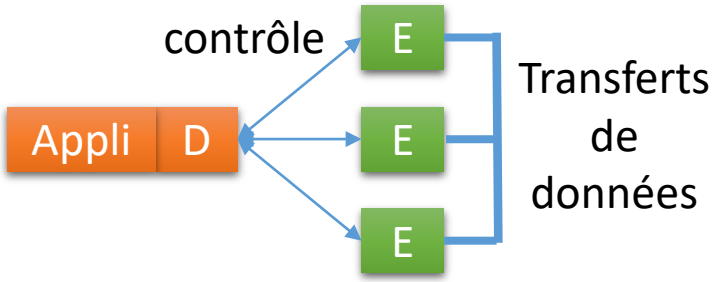
Architecture de la plateforme Spark

- Architecture répartie sur un cluster de machines. Deux types de machines
- 1 machine Master
  - Point d'entrée pour lancer une application
  - Attribution des ressources au démarrage de l'application
- Plusieurs machines Workers
  - ressources ram, cpu, comm



Architecture d'une application dans Spark

- Une machine pour le driver
  - Appli + contrôle de l'exécution
- Plusieurs Executors: service de calcul
  - 1 (ou plusieurs) executor par worker.



Gestion répartie des données

- Répartir = distribuer

- Répartir les données = partitionner + placer

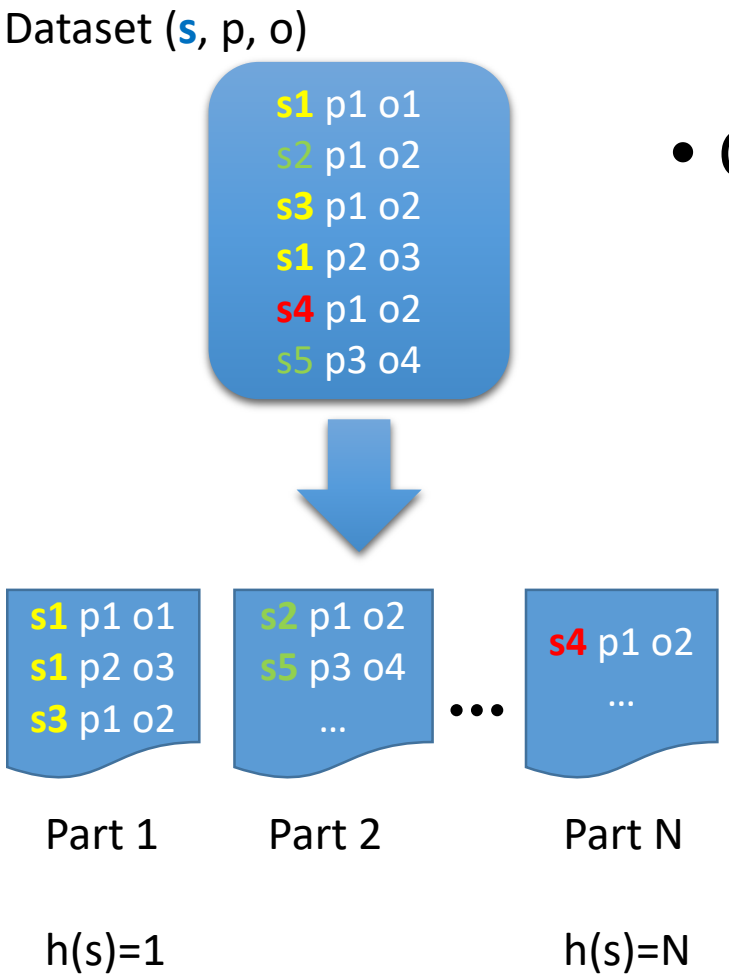
Partitionner les données

- Une collection est partitionnée en plusieurs morceaux appelés partitions.
  - Plusieurs éléments par partition
- Localité
  - Une partition est locale à une machine
  - Les éléments d'une partition sont ensembles sur une même machine
- Parallélisme
  - N partitions = N tâches indépendantes

Clé de répartition

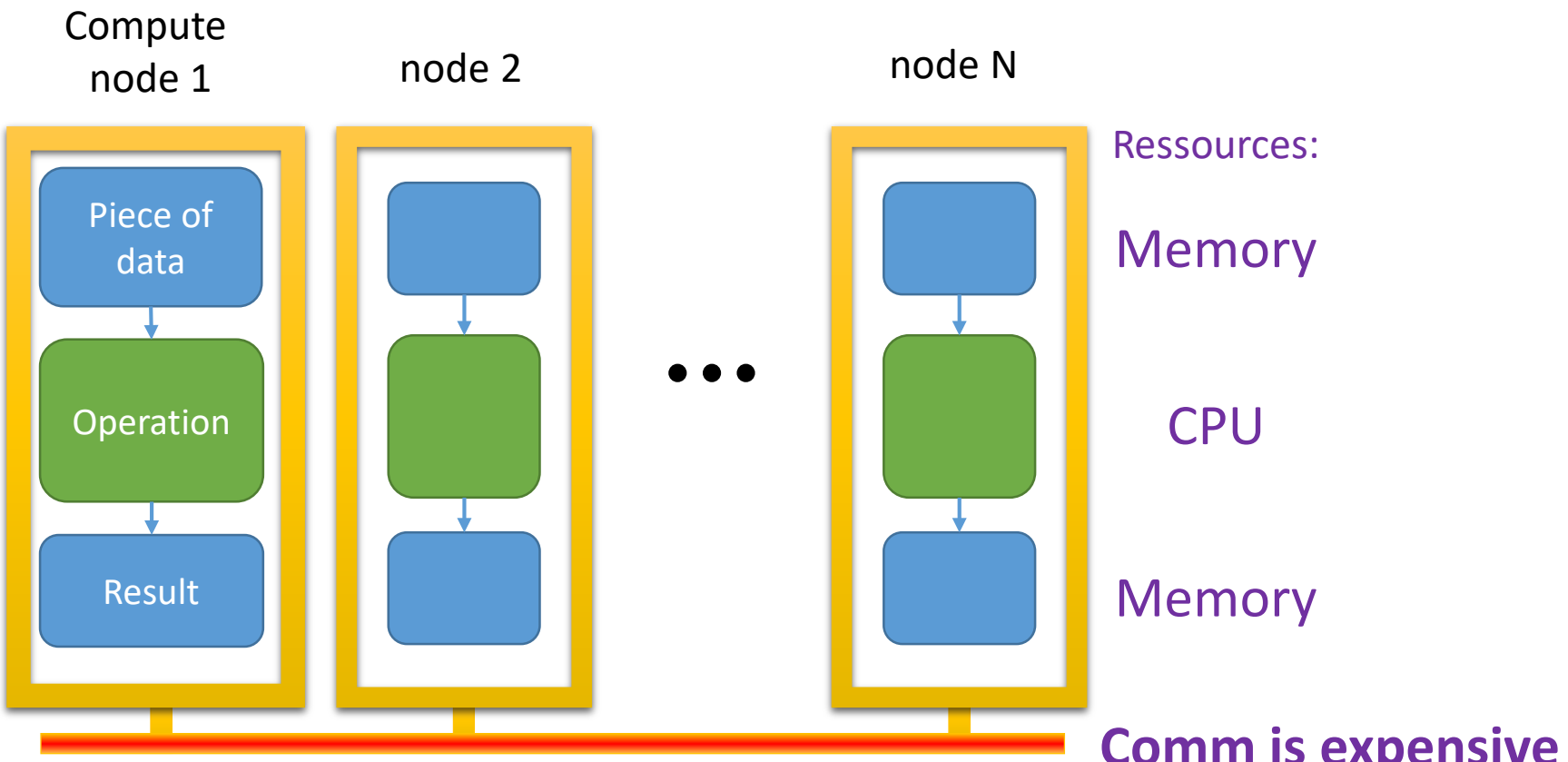
- Une collection C a une clé de répartition k noté C^k
- Par round robin: $C^{\emptyset}$
- Par hachage: C^x
  - h(x) → numéro de partition, avec h() fixée
  - Exple:
    - val A = USERS.repartition("ville"), noté A^{ville}
    - val B = USERS.groupBy("ville").count(), noté B^{ville}
- Par intervalle: $C^{seg(x)}$
  - C est triée selon x
  - seg(age) → numéro de segment du domaine de l'âges
    - Exple: val C = USERS.sort("age"), noté $C^{seg(age)}$

Répartition des données par hachage



- Clé de répartition = l'attribut **s**

Répartir les données sur les machines d'un cluster

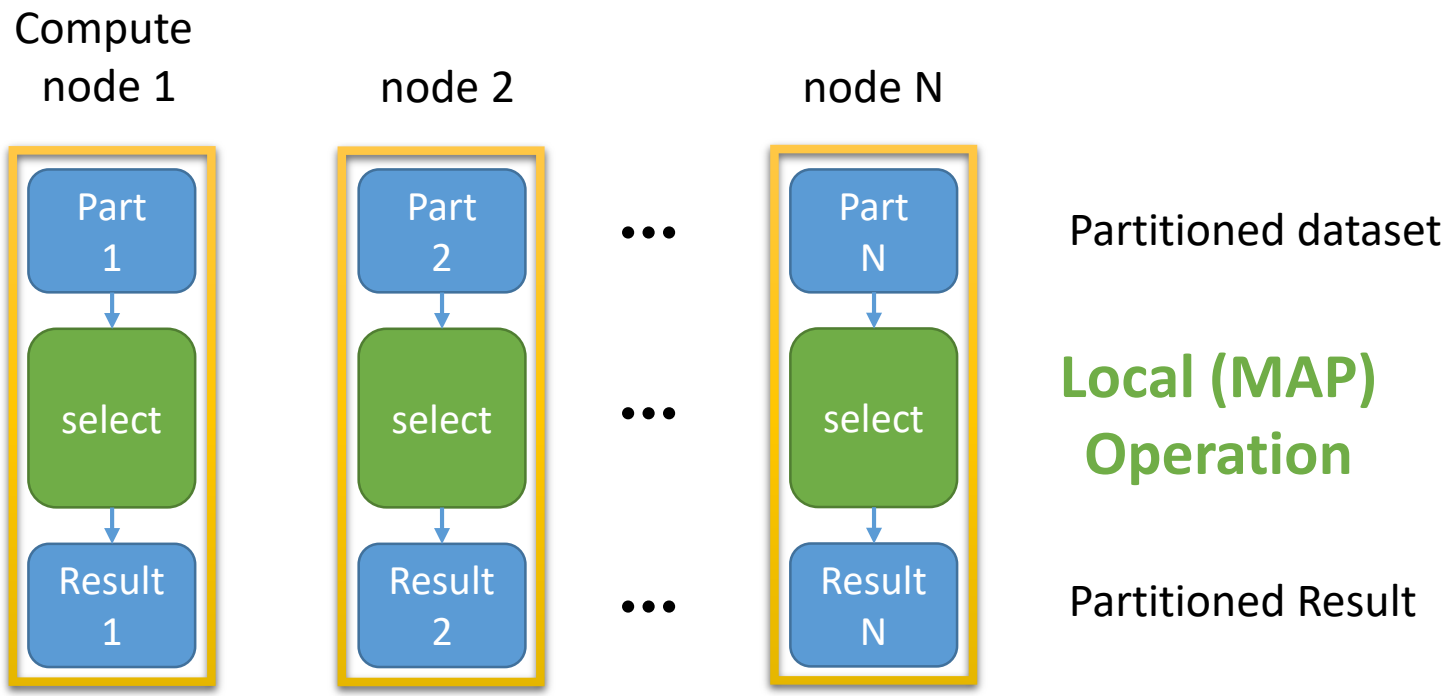


Opération locale

- Opération locale à un élément d'une collection
  - map, select
  - filter, where
- Opération locale aux éléments d'une seule partition
  - textFile
  - aggrégation
  - jointure sur la clé de répartition...

- Une opération locale **préserve** la clé de répartition
  - val User18 = USERS.where("age<18")
    - $USERS^{\varnothing} \rightarrow User18^{\varnothing}$
    - $USERS^{ville} \rightarrow User18^{ville}$

Exécution d'une opération locale

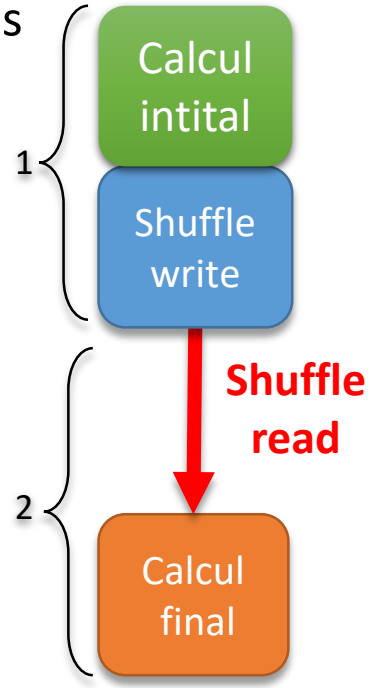


Opération globale

- Doit lire le résultat d'une opération locale évaluée sur plusieurs éléments d'une collection
  - ces éléments peuvent être dans différentes partitions
- Opération définie par

  1) un calcul initial préparant les données à envoyer

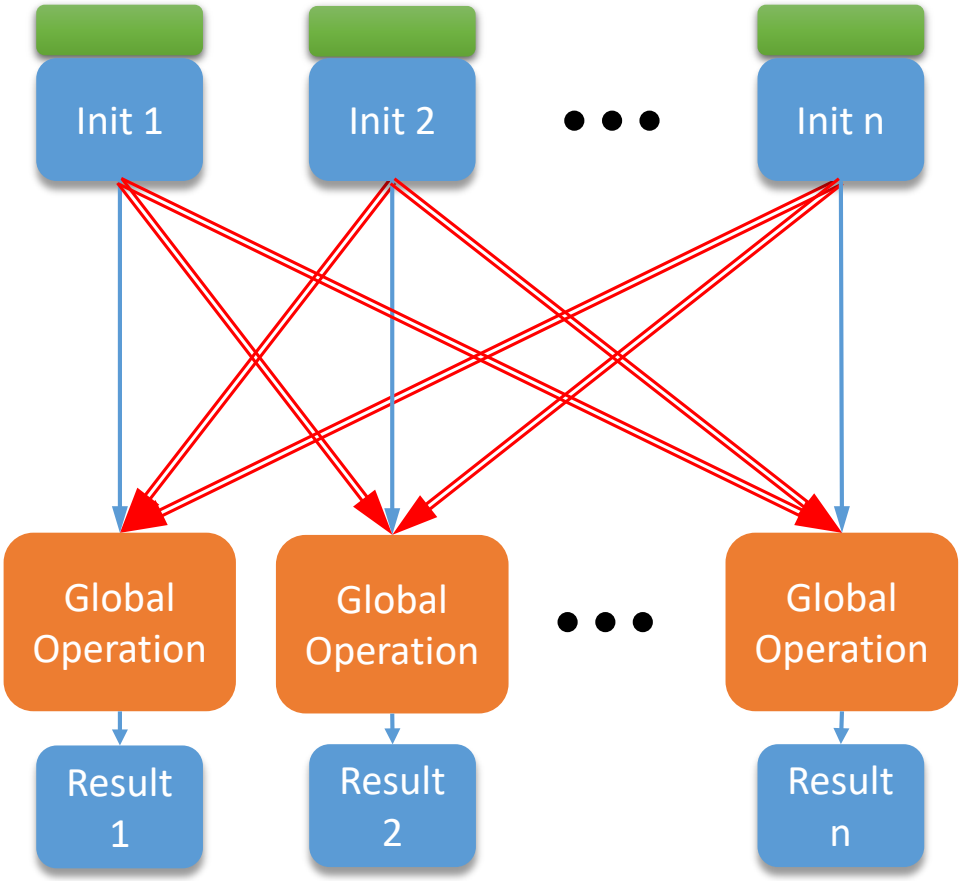
  2) un transfert des données servant au calcul final



Exemples

- reduce, reduceByKey
- groupBy, groupByKey, zipWithIndex
- join, cogroup
- sort, distinct, ...

Exécution d'une opération globale



Calcul initital

Shuffle write

Shuffle read

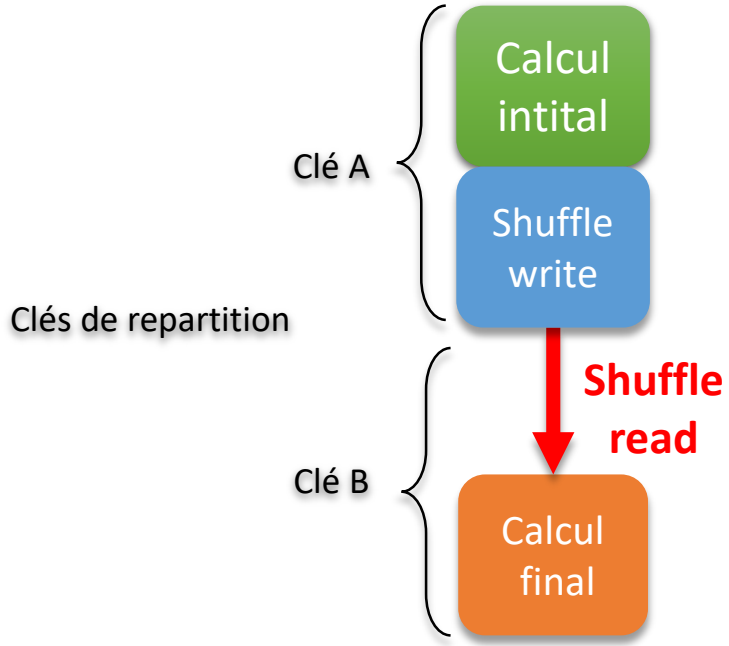
Calcul final

Changement de clé de répartition

- Une opération **globale**

modifie la clé de répartition

  - Donnée^{A} → Résultat^{B}



  - Exemple

    val NotesParis = USERS.where(ville="Paris").join(NOTES,numU)

    On a USERS$^{\varnothing}$, NOTES$^{\varnothing}$, et NotesParis^{numU}

Opération → Clé de répartition

Opération	Clé de répartition
A.repartition(att)	att
A.join(B, att)	att
A.groupBy(att)	att
A.distinct	A
A.dropDuplicates(att)	att
A.intersect(B)	A
A.reduceByKey(k)	k
filter, where	préservée
mapValue	préservée
map, select	perdue
A.mapPartition	perdue / préservée (user defined)

Transfert dépend des clés de répartition

- Ne pas re-répartir les données déjà "bien" réparties
  - Réduire les transferts
- Cas de la jointure
  - Répartition round robin par défaut des Users et Notes
    - val f1 = NOTES.join(USERS, "numU")
  - Notes déjà réparties
    - val N = NOTES.repartition("numU").persist() N.count()
    - val f2 = N.join(USERS, "numU")
  - Users déjà répartis
    - val U = USERS.repartition("numU").persist() U.count()
    - val f3 = NOTES.join(U, "numU")
  - Users et Notes déjà répartis
    - Val f4 = N.join(U, "numU")

Exécution d'une requête

- Requête
  - Composition d'opérations locales ou globales
- Plan : graphe d'opérations de transformation
  - Graphe orienté, acyclique et avec racine
  - La racine est l'**action** finale du plan
    - Exple : count, take, collect

Exécution d'un plan par étapes

- Un plan a plusieurs étapes
  - Plan=*Job* Etape=*Stage*
- Une étape
  - Bloc d'opérations **locales** consécutives
- Découpage du plan en étapes
  - Frontière: transfert compris dans une opération globale
    = **changement de la clé de répartition**
- Précédence entre les étapes
  - Début d'une étape
    - recevoir les données préparées par l'étape précédente
  - Fin d'une étape
    - préparer les données pour l'étape suivante

Exemple d'étapes

- Ex1
  - val a = USERS.where("age<30").select("ville").distinct()
- Ex2

val numExperts = AVIS.groupBy("numU").count().where("count >200").

select("numU")

  - numExperts.count

val experts = numExperts.join(USERS, "numU")

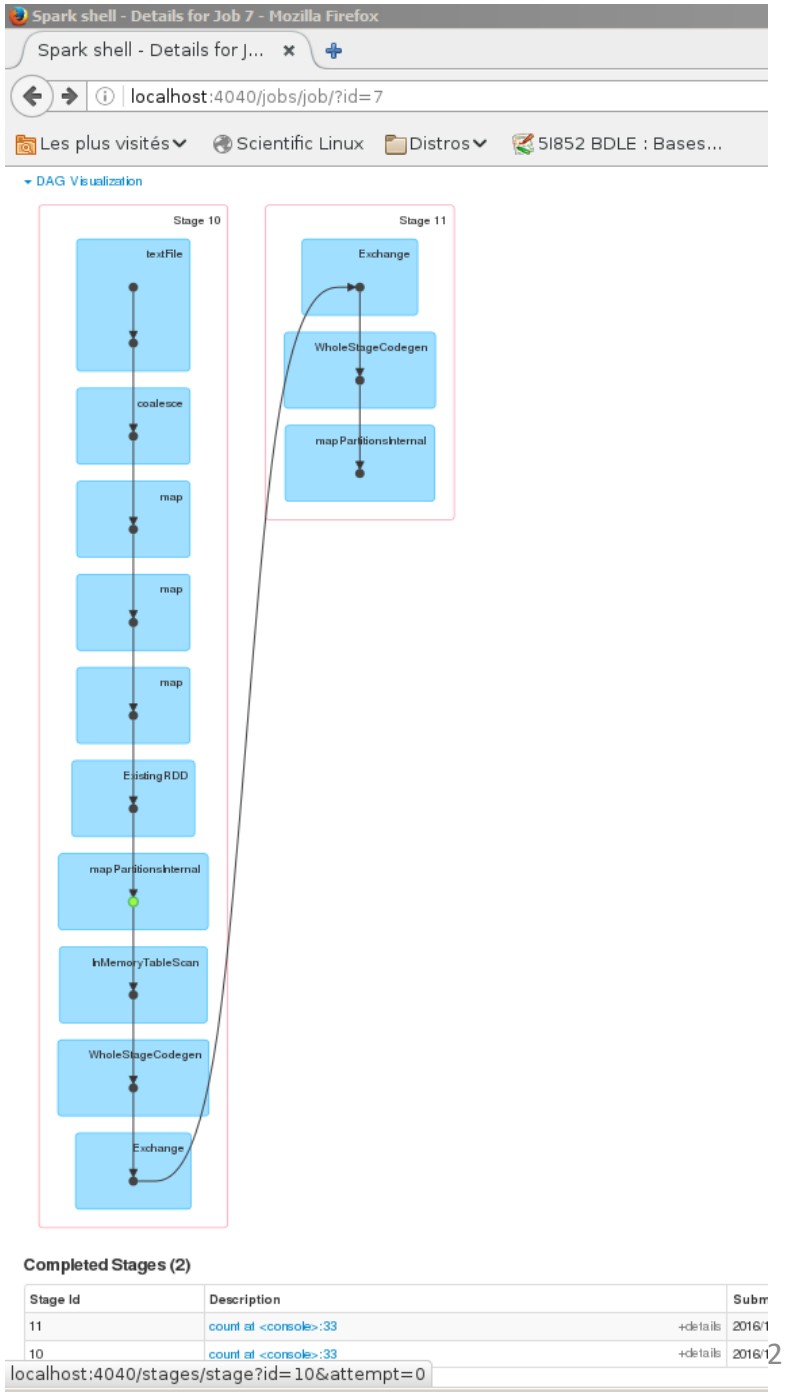
  - experts.columns
  - experts.count

Visualiser un plan d'exécution

- Interface graphique GUI:
  - URL localhost:4040
    - sauf avec databricks: url à déterminer
  - Etapes = blocs juxtaposés horizontalement
  - Transfert = arcs entre les blocs
- Résultats intermédiaires
  - Données persistantes
    - Une expression qui précède un point de persistance n'est **pas** ré-évaluée.
    - Persistance représenté par un noeud vert dans le graphe
  - Données transférées lors d'un shuffle read
    - Une expression qui précède un shuffle write n'est **pas** ré-évaluée si on ré-exécute la même requête une 2ème fois.
  - Expression non re-évaluée
    - Représentée en "grisé": skiped stage

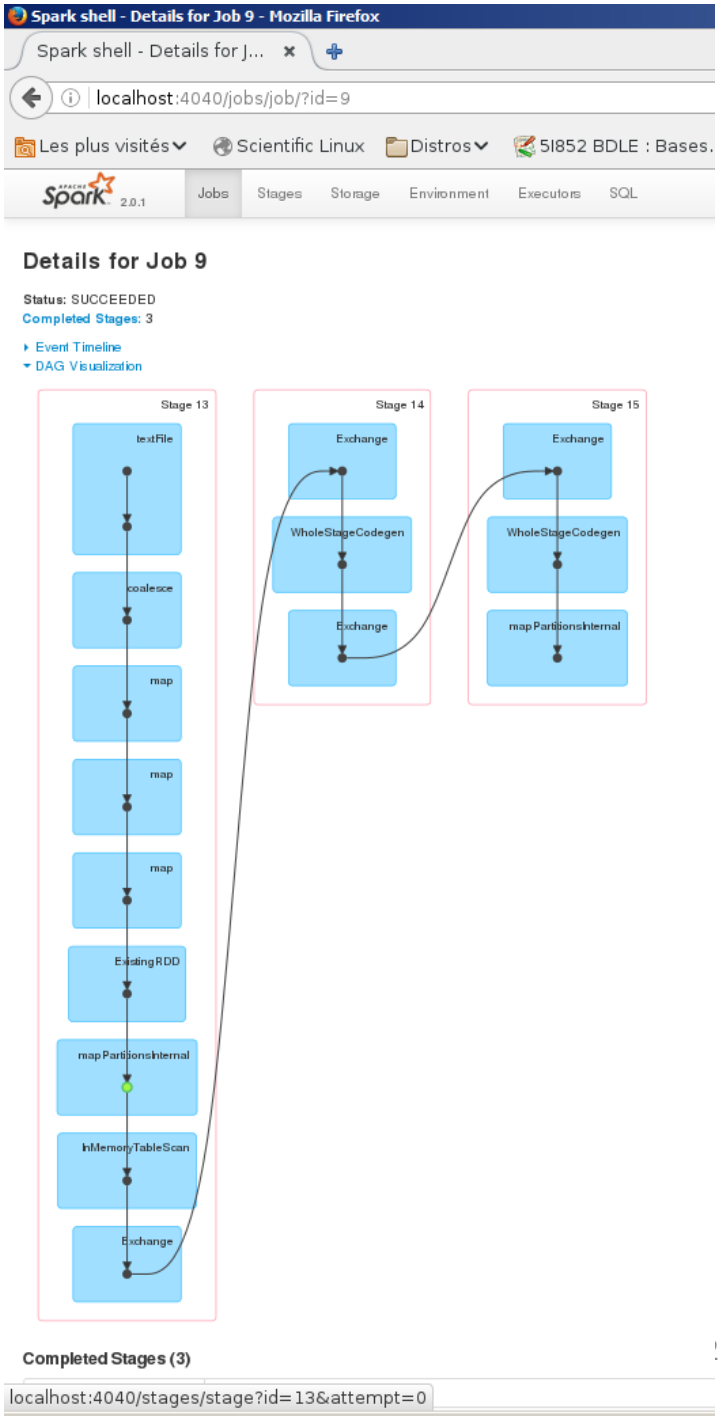
Exemple de plan

```
val a1 = AVIS.where("numF=1")
a1.count
```



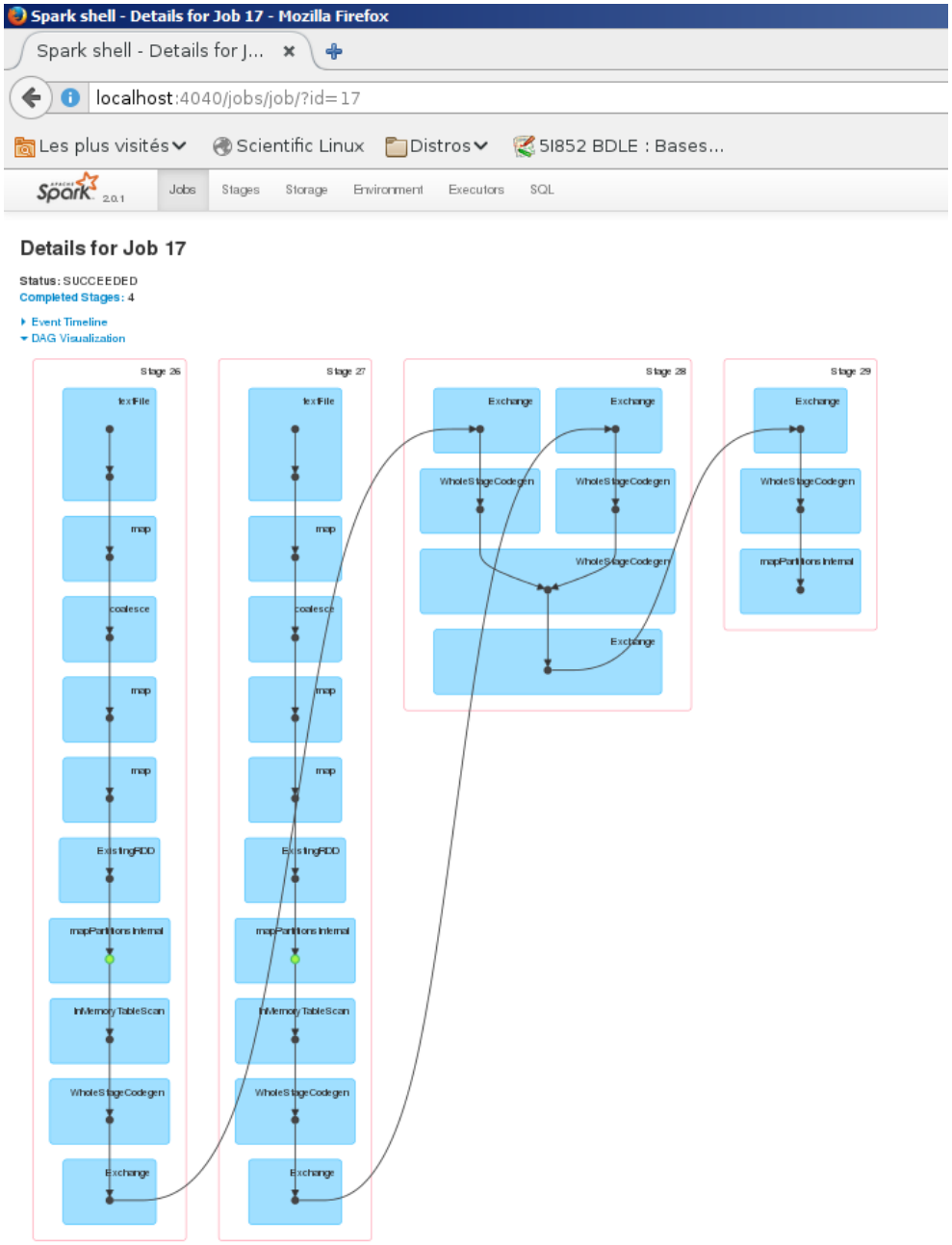
Exemple de plan

```
val s = AVIS.sort(col("note").desc)
s.count
```

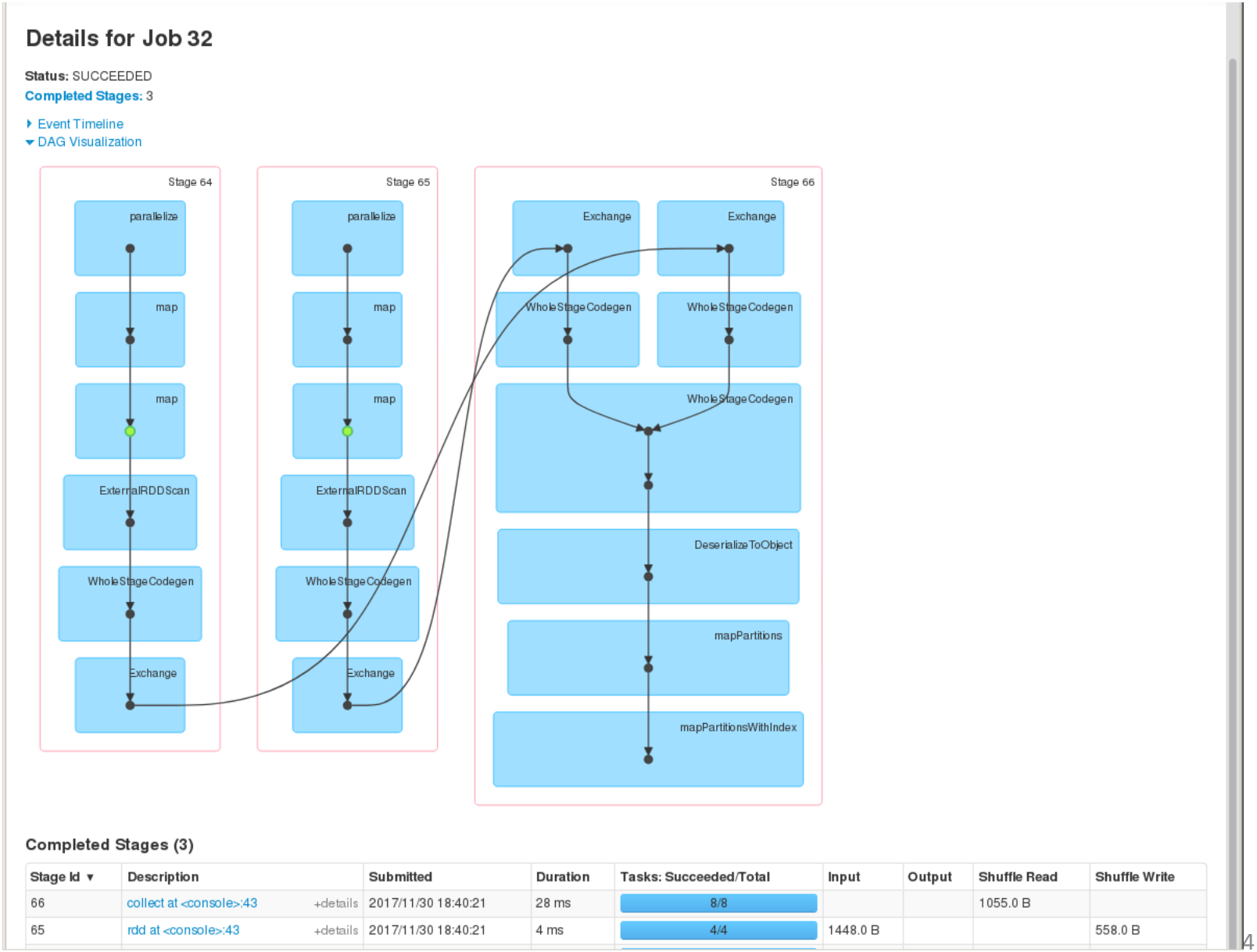


Exemple de plan

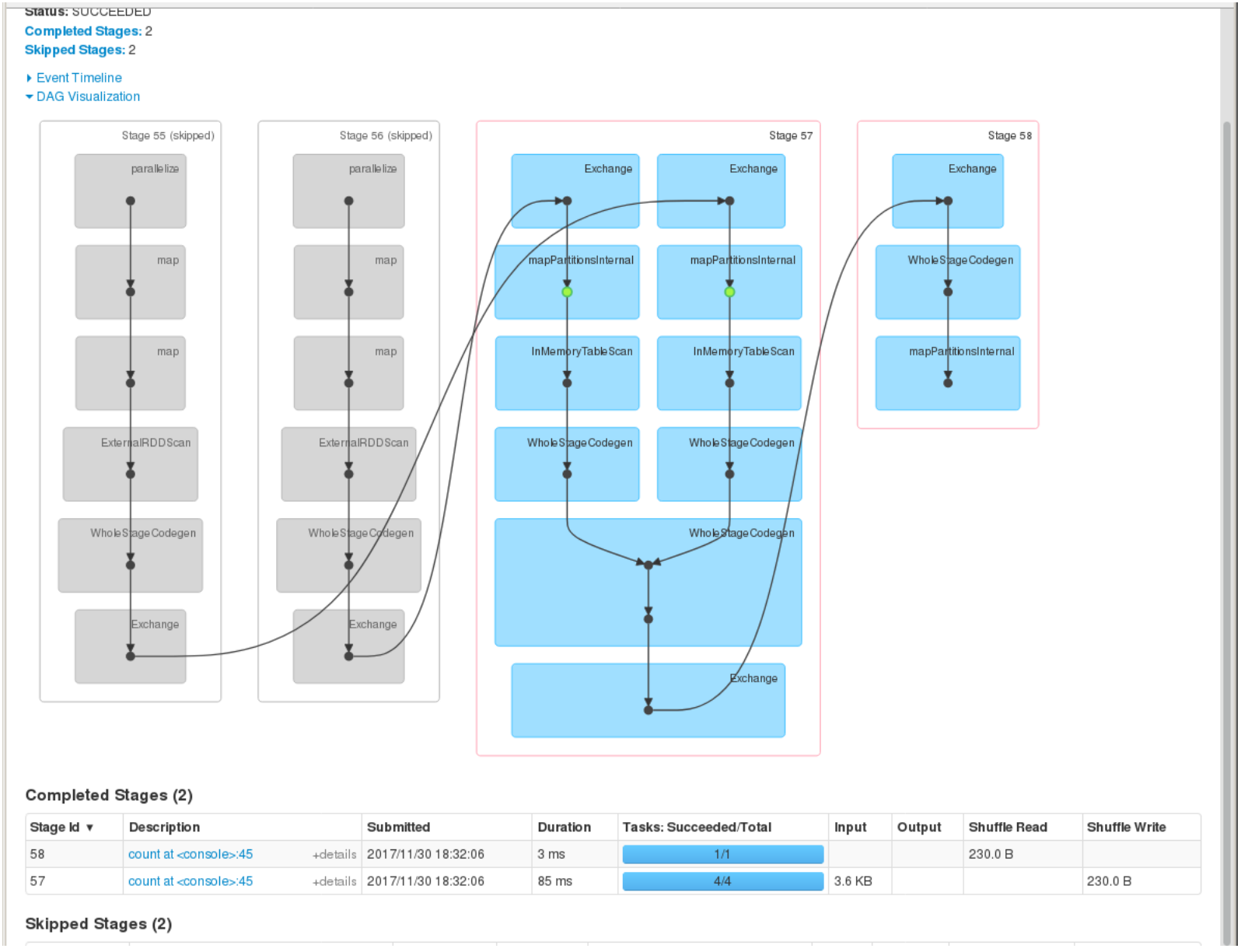
```
val t1 = TRIPLES.where("prop = '<book.book.characters>'")
t1.take(3).foreach(println)

val t2 = TRIPLES.where("prop = '<book.book.genre>'")
t2.take(3).foreach(println)

val j1 = t1.join(t2, "sujet")
j1.count
```



- Données partitionnées par round robin



- Données partitionnées par sujet (personne)



Diverses ref

- **Apache Spark: A Unified Engine for Big Data Processing**
  - https://vimeo.com/185645796
  - http://cacm.acm.org/magazines/2016/11/209116-apache-spark/fulltext
- Spark SQL under the hood – part I
  - Mikołaj Kromka, 2017
  - https://virtuslab.com/blog/spark-sql-hood-part-i/
    - Catalyst, dataframe, ...