UNIVERSITÉ PARIS SUD

Comprendre le monde, construire l'avenir®

université PARIS-SACLAY

Pourquoi tant d'engouement ?

- Parce-que c'est mieux que MapReduce
  - Plus simple d'utilisation pour un même problème
  - Plus performant sur les mêmes problèmes
  - Plus flexible, p.ex.: Algorithmes itératifs
- Parce-qu'il est orienté « data science »
  - Exploration interactive des données
  - Bibliothèque d'apprentissage
- Parce-que c'est à la mode
  - Ca compte !

Spécificités de SPARK vs MR

- Framework généraliste de calcul distribué
  - Enchaînement de traitements naturel
- « In memory » lorsque possible
  - Pas d'écriture intermédiaire entre opérations
  - Terasort : performance x30+
- Peut utiliser HDFS et YARN (MR2)
  - Également mode autonome

Exemple de Performances

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Jeux de données dans SPARK

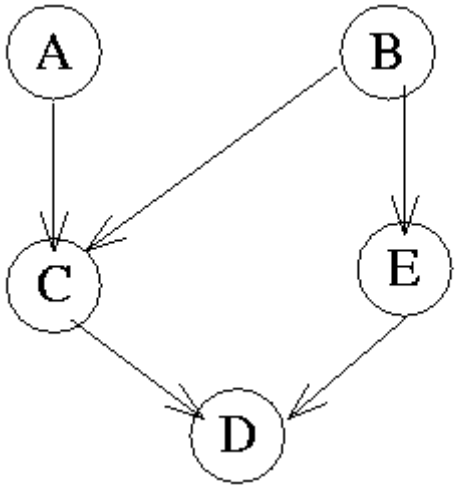
- Resilient Distributed Dataset (RDD)
  - Représentation d'un **ensemble** de données
  - Distribué (et redistribuable) sur les nœuds
  - Résistant : recalcul à la volée
  - En mémoire lorsque possible (sinon disque)
  - Traitement là où sont les données
- Dataframes (Spark 1.3)
  - données tabulaires (« RDD spécialisé »)
  - Lien avec SparkSQL

Traitement des données

- En entrée :
  - Fichiers : locaux, HDFS, S3 …
  - Bases de données : SQL, noSQL
  - Flux : MQ, Twitter, Kafka …
- Manipulation via transformations
  - RDD1 ⇨ RDD2 via transformation
    - Exemples : map, reduce, filtrage
- En sortie : fichiers, BDD, affichage, etc

Workflow Spark

- nœuds : transformations et actions
- Représentation sous forme de graphe
  - Graphe Acyclique Dirigé (DAG)
- Exécution lorsqu'actions
  - Affichage, écriture, collecte, …
  - « Exécution paresseuse »



Exemple de Workflow

1. Chargement de fichiers
2. Interprétation de chaque ligne
3. Filtrage des entrées incorrectes
4. Création de couples (clé, valeur)
5. Calcul valeur max par clé
6. Collecte et affichage

Exemple de Workflow

1. Chargement de fichiers ⇨ **textFile**
2. Interprétation de chaque ligne ⇨ **map**
3. Filtrage des entrées incorrectes ⇨ **filter**
4. Création de couples (clé, valeur) ⇨ **map**
5. Calcul valeur max par clé ⇨ **reduceByKey**
6. Collecte et affichage ⇨ **collect**

Exemple de Workflow

```
val ncdcFiles = sc.textFile("/ncdc/lite/*.gz")

val rdd1 = ncdcFiles.map(s => new NCDCData(s))
val rdd2 = rdd1.filter(data => (data.airTemperature!=9999))
val rdd3 = rdd2.map(data => (data.USAFID, data.airT° ))

val rdd4 = rdd3.reduceByKey((t1, t2) => math.max(t1, t2))

rdd4.collect().foreach(println)
```

Quelques transformations

- textFile, parallelize: Collection => RDD
- map(f) : E(x) => E(f(x))
- filter(f) : E(x) => E(x \ f(x) est vrai)
- union(RDD), intersection(RDD)
- distinct()
- groupByKey(), reduceByKey(func)
- aggregateByKey(U0)(f1, f2)
- sortByKey(bool: ascending)

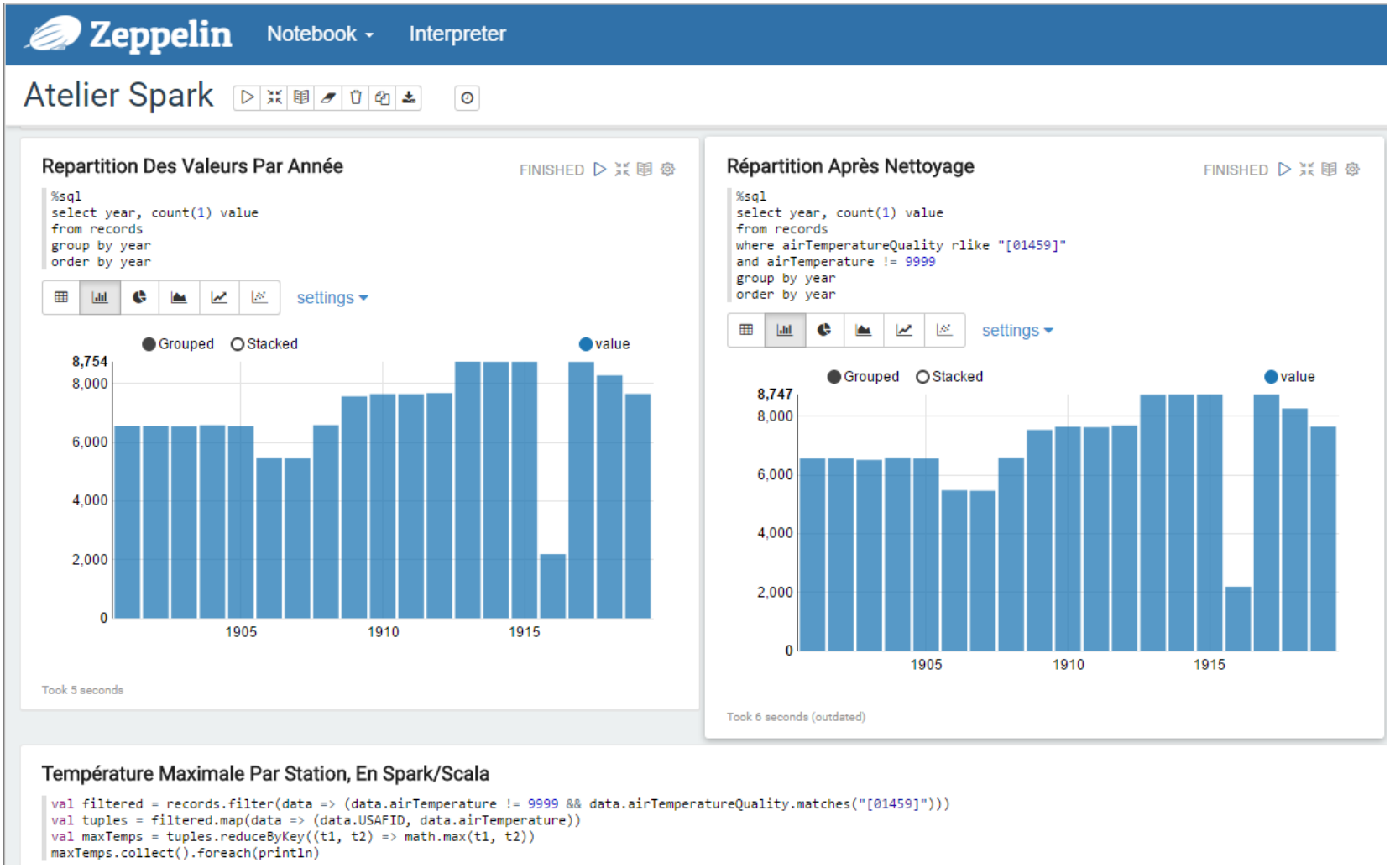
Quelques actions

- reduce(f)
- collect() : RDD => List
- first(), take(n), takeSample(n)
- count(), countByKey()
- saveAsTextFile(path)

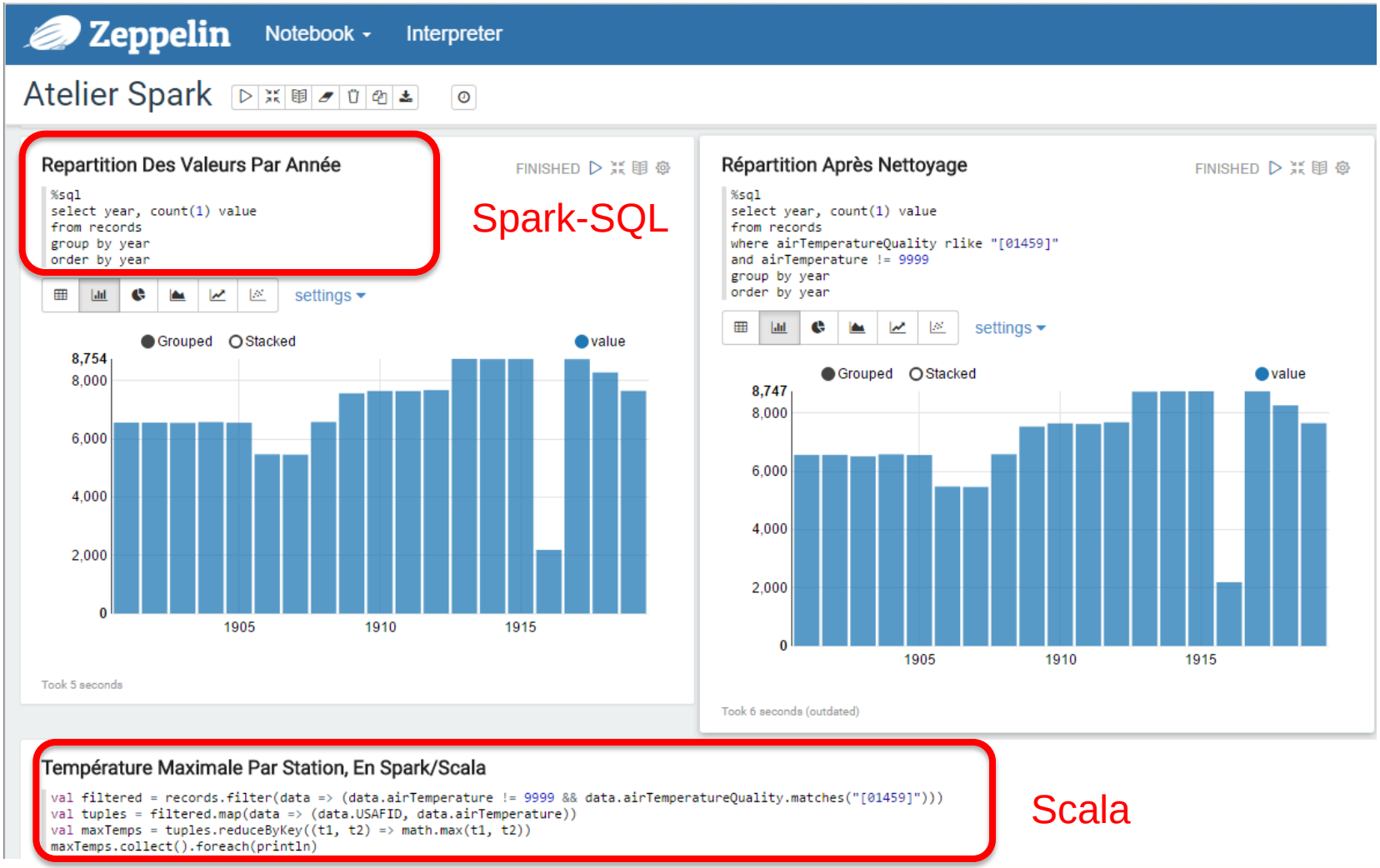
Comment programmer en Spark ?

- Langages : Java, Scala, Python, R
  - Programmation fonctionnelle
  - Scala est recommandé (natif)
- Possibilité de soumission de jobs
  - spark-submit
- Possibilité de console interactive (Scala)
  - spark-shell
- Notebooks
  - Zeppelin, Spark-Notebook

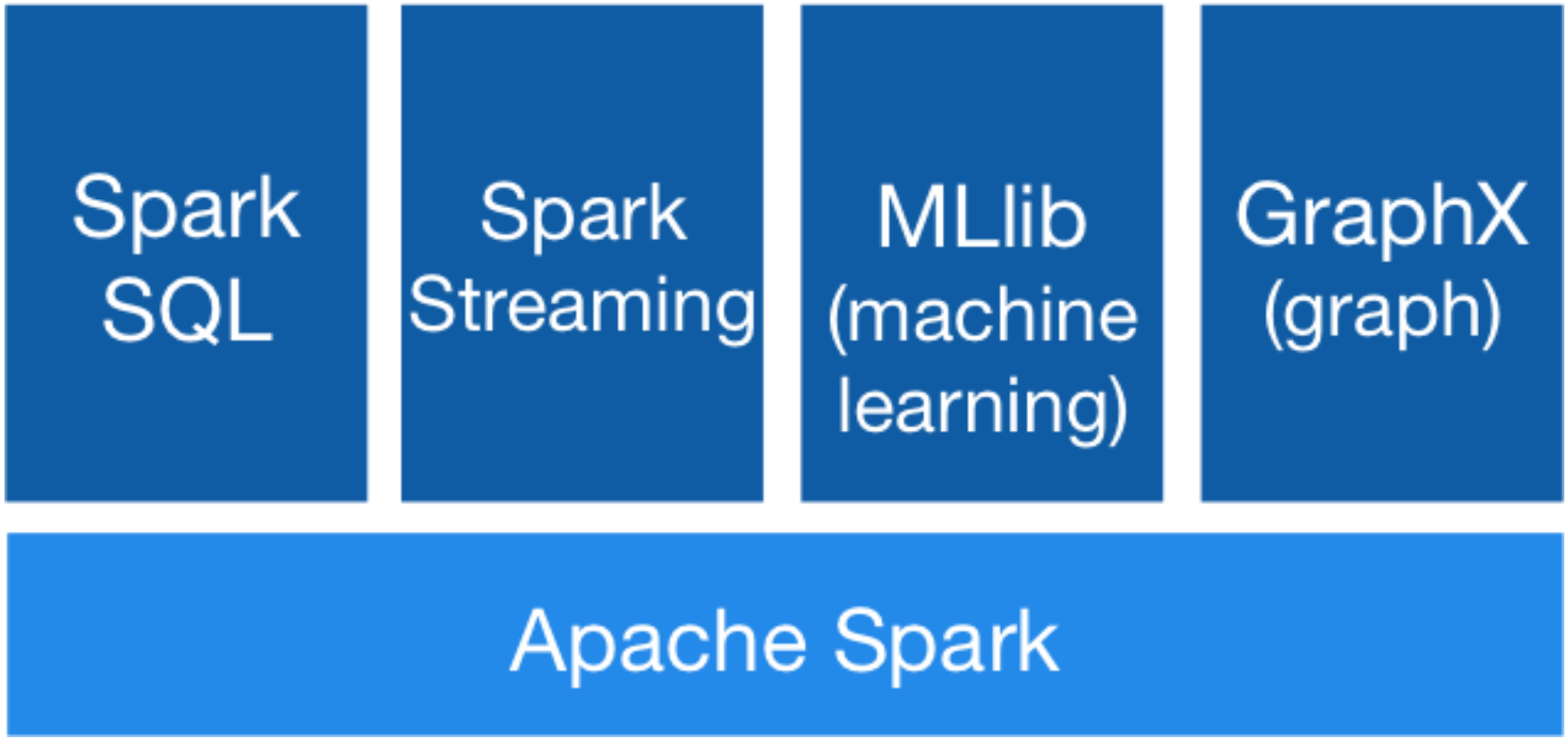
Zeppelin : un Notebook Spark



Zeppelin : un Notebook Spark



Bibliothèques Spark



Et le « small data » ?

- Distribution de petits jeux de données
  - Contrôle via parallelize() et partitions
  - P.ex.: Machine Learning
- Distribution des itérations
  - parallelize() encore, 2-4 partitions par CPU
  - P.ex.: SparkPi

Merci

- Questions ?