# Spark

## Introduction

• **Spark :** framework qui développé en réponse aux problèmes de Big Data

• **Problèmes adressés par Spark :**

— manipulation massive de données

— analyse en temps réel de données

— adaptabilité et scalabilité

• **Apache Spark :** framework Open Source de traitement de données

• **Histoire d’Apache Spark :**

— développé par l’université de Berkeley en 2009

— développé pour surmonter les problèmes des solutions existantes (Hadoop, MapReduce, etc.).

— repris par la fondation Apache en 2013

— vise à fournir une solution de traitement de données beaucoup plus rapide et plus facile à utiliser que celles existantes

• Spark est utilisé pour résoudre divers problèmes liés Big Data :

— analyse de données en temps réel

— analyse des objets connectés

— analyse bio-informatique et génomique

— analyse astronomique

— apprentissage automatique

— etc.

## Caractéristiques

### Distribué et scalable

• Spark est un système **distribué** qui traite les données sur plusieurs machines.

• **Cluster :** ensemble de ces machines qui communiquent entre elles

• **Nœud :** machine d’un cluster

• Il est également facilement **scalable** et **flexible** : il permet de :

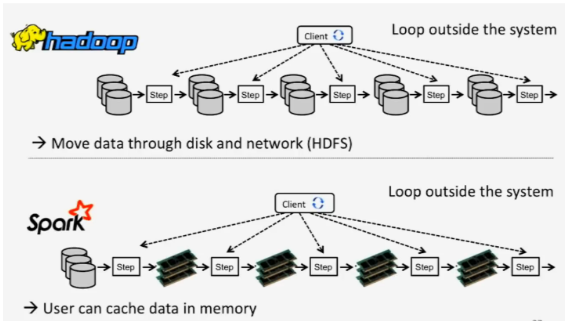
— ajouter des nœuds pour augmenter la capacité de traitement

— supprimer des nœuds inutilisés pour réduire les couts

### Rapide et performant

• Spark effectue le traitement des données **en mémoire**: les données sont stockées et manipulées directement dans la mémoire vive de l’ordinateur.

• Cela permet un traitement très rapide des données sans avoir besoin de les écrire sur un support entre chaque étape.



• Spark utilise le moteur d’exécution haute performance DAG (Directed Acyclic Graph) pour accélérer le traitement des données parallélisées.

### Facile d’utilisation

• Spark est conçu pour être facile à utiliser.

• Il propose des API simples et intuitives.

• Il est supporté dans plusieurs langages de programmation, dont Python, Scala, Java, R, SQL, etc.

• Il est possible d’utiliser Spark avec des notebooks comme Jupyter ou dans un terminal interactif via l’outil **spark-shell**.

### Limites de Spark

• **Configuration** **complexe** des clusters qui demande une connaissance en administration système avancé

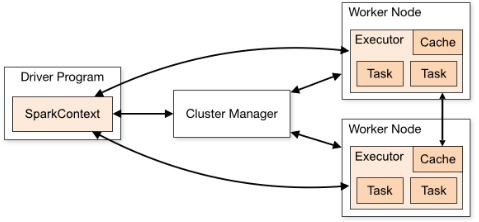
• **Consommation de mémoire de travail importante :** le stockage et traitements en mémoire vive par Spark peut être assez gourmands en mémoire

• **Latence :** le transfert de données d’un nœud à l’autre peut être long et ralentir les traitements itératifs

## Architecture

• Spark est un système de traitement de traitement distribué.

• Il nécessite de nombreux composants pour fonctionner. Ils sont réunis dans un cluster



• Le travail de Spark se base sur une architecture de type **maitre-esclave :**

— le maitre est le point d’entrée de l’application Spark

— les esclaves sont les machines de traitement

• Dans cette architecture, on trouve : le driver, les tâches à accomplir, les workers, le cluster manager, le contexte, etc.

### Le Driver

• C’est le point d’entrée de l’application Spark.

• Le nœud driver est le coordinateur de travail à effectuer.

• C’est lui qui créé et détient le contexte Spark responsable de la soumission des tâches aux workers.

• Note : sorte de chef qui planifie et suit l’exécution des tâches.

### Les Workers

• Ce sont les nœuds machines qui effectuent le travail.

• Un worker possède une mémoire dédiée au stockage des données et des partitions de travail.

• C’est lui qui effectue le travail d’une tâche à accomplir.

### Le Cluster Manager

• C’est le gestionnaire de ressources du cluster.

• Le cluster manager est responsable de gérer l’état des workers, la répartition et la planification des tâches.

• Spark peut utiliser différents cluster manager :

— Spark Standalone (manageur local)

— Apache Mesos (manageur distribué)

— Hadoop YARN (manageur distribué)

— Kubernetes (manageur distribué)

### Le contexte

• C’est l’interface entre le driver et les workers.

• Il permet de **paramétrer le cluster** et de soumettre les tâches aux workers avec l’aide du cluster manager.

• Le contexte est le point d’entrée qui récupère les données à traiter et le travail à effectuer.

### Une tâche

• C’est une **unité de traitement** de données.

• Une tâche est créée par le driver à partir des données sources et du programme à exécuter.

• L’exécution d’un programme est donc une suite de tâches à effectuer sur une suite de données.

### A

• Le **driver** reçoit les données et le traitement.

• Le **contexte** définit le cluster et les tâches à effectuer.

• les tâches sont réparties sur les workers sous la supervision du **cluster manager**.

• Les **workers** effectuent les tâches et retournent les résultats au driver.

## Fonctionnement

• Le principe de travail de Spark est de faire du traitement de données massivement et rapidement.

• Pour réaliser cet objectif, Spark utilise la **parallélisation**.

### Parallélisation

• La parallélisation permet de **diviser** un problème en plusieurs sous-problèmes.

• Puis de les résoudre en même temps (parallèle) sur différents processeurs.

• Le plus souvent réparti sur plusieurs machines : le cluster.

• Cette **parallélisation** est réalisée par :

— la division des données en Resilient Distributed Dataset (RDD)

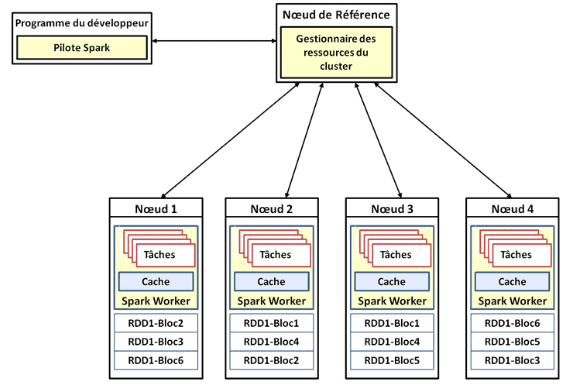
— la division du programme par le Direct Acyclic Graph

### Resilient Distributed Dataset (RDD)

• Pour améliorer le traitement des données, Spark manipule les informations sous un format divisé en blocs d’informations réparties sur les différents nœuds.

• C’est le Resilient Distributed Dataset (RDD).

• Les blocs de données sont appelés **partitions** et sont stockées en plusieurs exemplaires sur les différents nœuds du cluster.

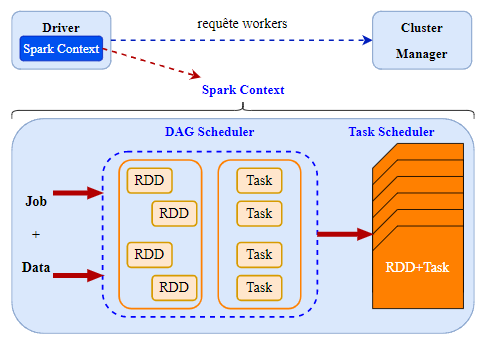


### Direct Acyclic Graph (DAG)

• Lorsque Spark reçoit un programme, celui-ci va être également transformé.

• Le DAG va permettre de **diviser le travail** à accomplir en action à effectuer les unes à la suite des autres.

• Une tâche est ensuite planifiée pour une action sur un bloc de données



### Résumé

• Spark va diviser les données en partitions et les répartir sur les différents nœuds du cluster.

• Puis va diviser le programme en tâches à effectuer sur les partitions.

• Ensuite, il va répartir les tâches sur les différents nœuds du cluster.

• Le travail peut alors être effectué en même temps.

### Spark : flux de données

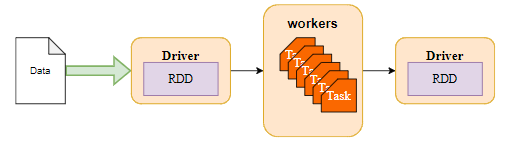
• Le travail de traitement des données par Spark peut se faire selon différentes configurations :

— traitement par lots (batch)

— traitement en micro-batch

— traitement en temps réel (streaming)

### Traitement par lots (batch)



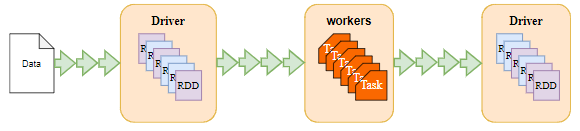
• Consiste à charger en mémoire l’ensemble des données à traiter.

• Une seule RDD sur les différents nœuds du cluster (stocké et traité de manière distribuée).

• Le traitement s’effectue une fois le chargement complet des données effectuées.

• L’ensemble est ensuite regroupé sur un seul RDD.

### Le traitement en micro-batch



• Le traitement en micro-batch consiste à charger les données par petits lots appelés **micro-batch** au lieu de tout charger d’un coup.

• Intérêt : traitement au fur et à mesure par petits groupe sans attendre d’avoir tout récupéré.

• Le traitement en micro-batch est particulièrement adapté aux données très volumineuses ou aux données incrémentales.

### Le traitement en temps réel (streaming)

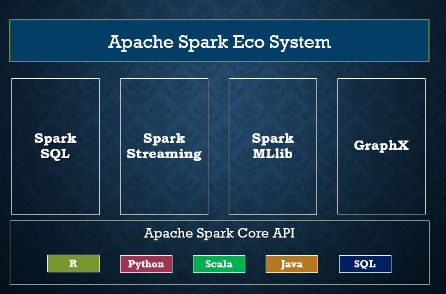
• Le traitement en temps réel consiste à traiter les données au fur et à mesure de leur arrivée.

• C’est le traitement le plus rapide car les données sont traitées au fur et à mesure de leur arrivée.

• En pratique, le traitement en temps réel est une agrégation des données traitées en micro-batch en continu.

## Modules Spark

• Spark est subdivisé en différents modules répondant chacun à un besoin spécifique :



### Spark Core

• Module qui fournit les fonctionnalités de base de Spark.

• Fournit l’ensemble des fonctionnalités permettant de :

— définir les inputs / outputs du job Spark

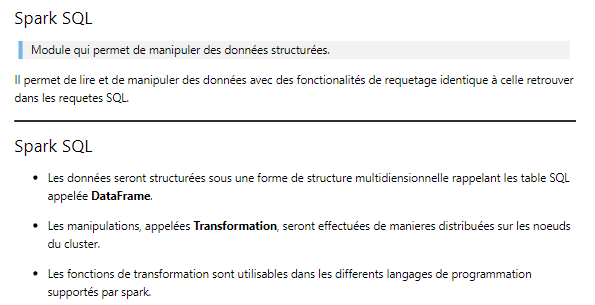
— manipuler et transformer les RDD

— lancer des jobs Spark

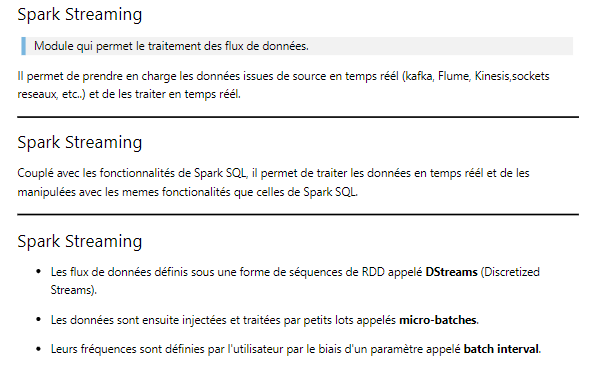
— établir le contexte

• Il contient toutes les API permettant de piloter ces fonctionnalités dans les différents langages de programmation supportés par Spark : R, Python, Scala, Java, SQL

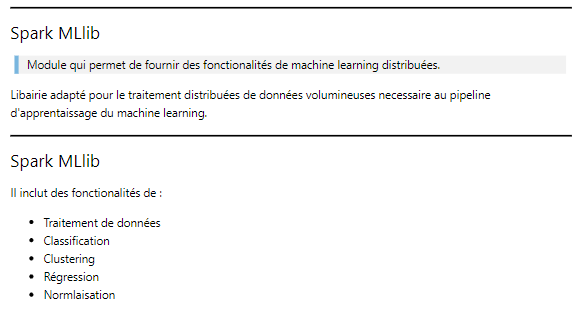
### Spark SQL



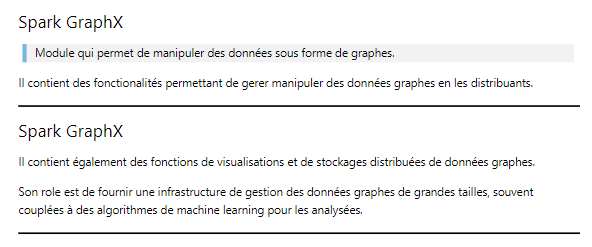
### Spark streaming



### Spark MLlib



### GraphX



## Session et contexte

## RDD et Dataframe

## Utilisation RDD

## A

A

•

—