Université Mohammed V- Rabat
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique
Filière Génie Informatique et Digitalisation

Traitement Big Data

Pr. N. EL FADDOULI
nfaddouli@gmail.com

2023-2024

Chapitre 2: Architecture Spark

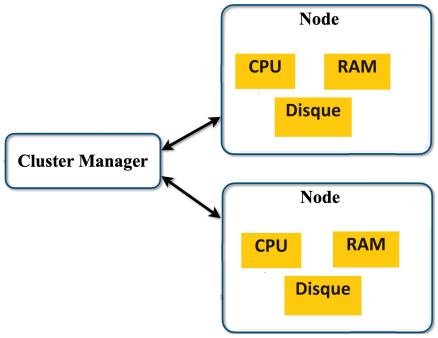
- ❑ Modèle d'application
- ❑ Cluster Spark et Gestionnaire de cluster
- ❑ Les composants de l'architecture Spark: Driver/Executors
- ❑ Spark en local.
- ❑ Interaction Client-Spark

Architecture d'Apache Spark: Modèle d'application

- Dans **MapReduce**, l'unité de calcul de plus haut niveau est le **Job**; dans Spark, l'unité de calcul de plus haut niveau est l'**application** qui ***génère un ou plusieurs Jobs***
- Dans un Job MapReduce, on **charge les données**, et on leur **applique les fonctions Map**, **Shuffle** et **Reduce**, et on **enregistre les résultats sur HDFS**. Dans Spark, une **application est une entité autonome** qui exécute le code de l'utilisateur et envoie les résultats du calcul.
- **Spark** exécute les applications en utilisant les ressources coordonnées de plusieurs nœuds.
- Contrairement à **MapReduce**, qui **démarre un nouveau processus pour chaque tâche** (Map ou Reduce), une **applications Spark** est exécutée via des **processus indépendants** sous la coordination de l'objet **SparkSession** (ou **SparkContext**) généré et utilisé dans le programme pilote (**Driver**).
- Les applications **Spark** peuvent **maintenir des processus en cours d'exécution en leur nom** dans les nœuds du cluster **même lorsqu'aucune tâche n'est en cours d'exécution**, et **plusieurs applications peuvent lancer des tâches sur le même nœud**.

Architecture d'Apache Spark: Le Nœud Maitre

- **Spark** fonctionne en cluster avec une architecture **Maitre/Esclave:** un seul nœud maitre et plusieurs nœuds esclaves.
- **Nœud maitre** : le nœud sur lequel le **gestionnaire de cluster** Spark est exécuté.



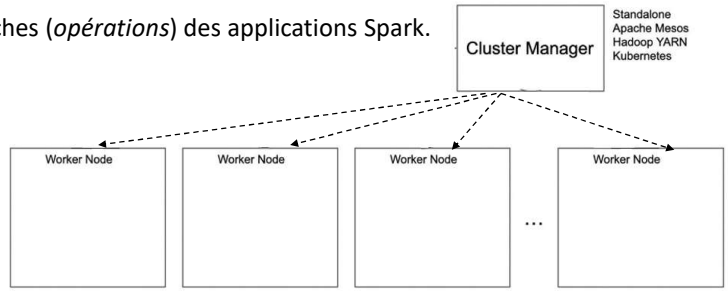
- **Exemple**: Dans **YARN**, le gestionnaire du cluster est le **Ressource Manager**.

Architecture d'Apache Spark: Types de gestionnaires de cluster

- Le nœud maître du cluster Spark est l'un des deux éléments, selon le gestionnaire de cluster utilisé :
  - **Gestionnaire intégré** en mode **Standalone**: Lorsque Spark utilise son propre gestionnaire de cluster intégré, le nœud maître est appelé le "**Spark Master**". Il est responsable de la **gestion des ressources du cluster**.
  - **Gestionnaires de cluster externes:** Lorsque Spark est déployé sur des **gestionnaires de cluster** comme Apache Mesos, YARN ou Kubernetes, le **nœud maître est géré par le gestionnaire de cluster** lui-même. Dans ce cas, le gestionnaire de cluster (par exemple, le **Resource Manager** pour YARN) joue le rôle de **nœud maître** pour la **gestion des ressources**, tandis que **Spark** utilise ces ressources selon les besoins pour **coordonner l'exécution des applications.**

Architecture d'Apache Spark: Les nœuds Workers

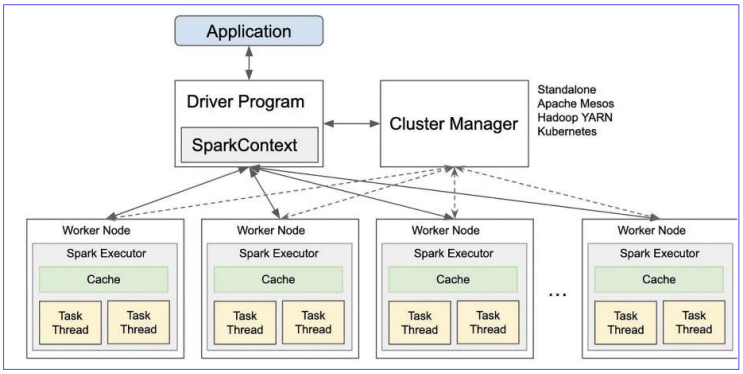
- Un nœud esclave, appelé **Worker Node**, héberge des ressources (*conteneurs*) nécessaires pour l'exécution des tâches (*opérations*) des applications Spark.



- Dans un cluster **YARN**, le **Worker** est le **Node Manager**. Il s'agit d'un processus qui s'exécute sur chaque nœud du cluster et gère les ressources (CPU, RAM) de ce nœud. Il est responsable de l'attribution de ces ressources aux conteneurs.
- Sur chaque **Worker Node**, on peut lancer au démarrage d'une application un ou plusieurs **processus** qui s'exécutent pendant toute la durée de vie de l'application.
- Ces processus seront chargés d'exécuter les **tâches** de l'application.

Architecture d'Apache Spark: Driver/Executors (1/3)

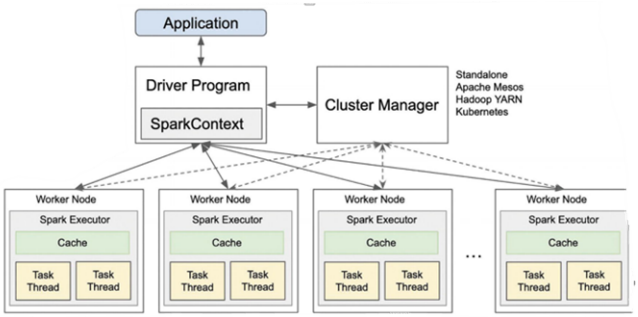
- Une **application** (*code utilisateur*) est **exécutée** par un **Driver Program spécifique** qui est un **processus dédié** pour l'application (***un Driver par application***).
- Le **Driver** génère un objet **SparkContext** pour communiquer avec le **gestionnaire du cluster** et les **Workers**.
- Le **Driver** obtient les ressources nécessaires sur des **Workers** (*par exemple: des **conteneurs** de YARN*) pour exécuter les tâches de l'application.



- Le **Driver** lance des **Executors** sur les nœuds **Workers** et leur envoie le code de l'application.

Architecture d'Apache Spark: Driver/Executors (2/3)

- Le **Driver** passe les tâches de l'application aux **Executors**, par exemple: **filtrer** des données, …
- Chaque **tâche** est lancée dans un **thread** séparé.
- Les **Executors** envoient des battements de cœur (**Heartbeats**), par défaut toutes les 10s, au **Driver** qui fait le suivi de l'état d'avancement des tâches exécutées par chaque **Executor**.
- Les **Executors** peuvent mettre en **cache** des jeux de données pour accélérer les applications qui y accèdent à plusieurs reprises (***itérations***).
- Sur chaque **Worker node**, on aura plusieurs **Executors** des différentes applications en cours d'exécution.



Architecture d'Apache Spark: Driver/Executors (3/3)

- Le nombre d'**Executors** peut varier en fonction de la configuration spécifique de l'application Spark et du cluster.
- On peux ajuster le nombre d'**Executors** en fonction des besoins en matière de traitement parallèle et de l'utilisation des ressources du cluster.
- Le nombre d'**Executors** est principalement contrôlé par deux propriétés de configuration :
  - **spark.executor.instances** spécifie le nombre d'**Executor**s à allouer à l'application.
  - **spark.executor.cores**: spécifie le nombre de cœurs CPU à allouer par **Executor**.
- Le nombre total d'**Executors** à allouer dépend des ressources disponibles du cluster.
- Les **Workers** à utiliser par un **Driver** sont déterminés par le **gestionnaire du cluster**.
- Le choix des **Workers Nodes**, par le gestionnaire du cluster, pour lancer les **Executors** dépend des ressources disponibles et de l'emplacement des données à traiter sur les nœuds du cluster.

  ⇨ Minimiser le transfert de données sur le réseau.

Architecture d'Apache Spark: Exécution en local

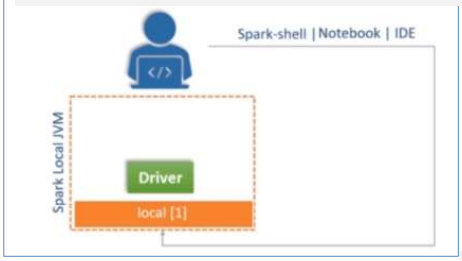
- Exécution de **l'application Spark localement** (*une seule machine*) et **non sur un cluster**
- Cela est utile pour le **développement** et le **débogage** sans avoir besoin d'un cluster complet.
- On indique le **mode à utiliser** lors de la création de l'objet **SparkContext** ou **SparkSession** en précisant le nombre de cœurs CPU virtuels à utiliser :

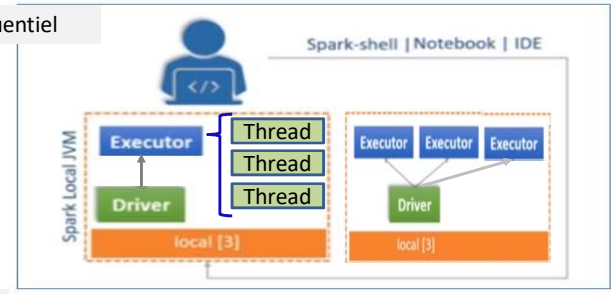
Exemples en Python:

```
from pyspark import SparkContext
sc = SparkContext("local[1]", "Hello Spark")
```

```
from pyspark.sql import *
spark = SparkSession.builder \
        .appName("Hello Spark") \
        .master("local[3]") \
        .getOrCreate()
```

Driver avec un cœur CPU ➔ Traitement séquentiel



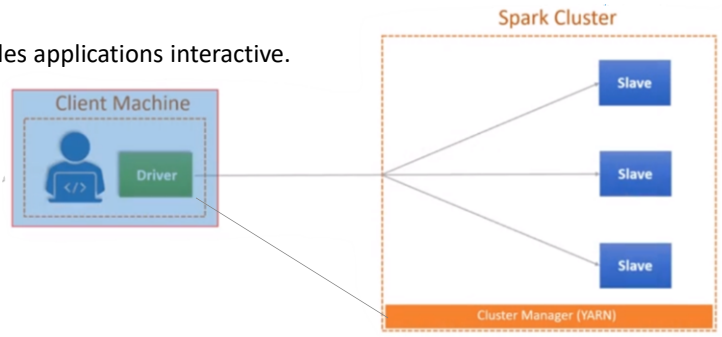


Executor avec 3 cœurs CPU ➔ Traitement parallèle

Architecture d'Apache Spark: Interaction Client-Spark

Mode Client

- Le **Driver** est lancé sur le **nœud client** en **dehors du cluster**, qui est généralement la machine où on exécute la commande pour lancer l'application Spark, par exemple avec **Spark-Shell**
- Mode principalement utilisé pour le développement et le débogage.
- On doit s'assurer que les dépendances de l'application Spark sont disponibles sur le nœud client.
- Ce mode est utilisé dans les applications interactive.



Architecture d'Apache Spark: Interaction Client-Spark

Mode Cluster

- L'ensemble de l'application Spark, y compris le **Driver**, est soumis à l'exécution sur le cluster Spark lui-même, par exemple avec la commande **spark-submit**.
- Le **Driver** s'exécute sur un nœud du cluster.
- Ce mode est utilisé pour les charges de travail de **production** et les gros traitements Spark, car il utilise efficacement les ressources du cluster.