

Apprendre le Big Data par la pratique avec Spark



Le Guide du Data Engineer pour maîtriser Spark

LES EDITIONS JUVENAL & ASSOCIES

Plan du livre

INTRODUCTION.....	4
1- Spark : le moteur Big Data 4-en-1	6
2- LES RDD : LA CLE DE VOÛTE DE LA PERFORMANCE DE SPARK	11
2.1. Comprendre les RDD.....	12
2.2. Utiliser les RDD de Spark	14
3- PRATIQUE DE SPARK !.....	17
3.1. Installer Spark sur son PC	17
3.2. Créer une session Spark.....	20
3.3. Savoir manipuler les RDD	22
3.4. Les Data frames : la clé de voûte de Spark SQL.....	23
<i>Création d'un data frame.....</i>	<i>24</i>
4. EXECUTION DES RDD DANS UN CLUSTER	25
Le lignage de RDD	26
CONCLUSION	29
QUIZZ DE VALIDATION DE COMPETENCES	30
Votre avis compte !.....	34

Mentions légales

Cet ebook est une propriété exclusive de **Juvénal CHOKOGOUE**, le représentant légal des Editions Juvénal & Associés, et a fait l'objet d'un dépôt légal. Toute personne a le droit de le télécharger et de l'utiliser uniquement sous les conditions listées ci-dessous :

- Vous avez le droit de copier ou d'intégrer partiellement le texte de l'ebook dans vos propres travaux à condition de mentionner le nom de l'auteur, donc Juvénal CHOKOGOUE ;
- L'intégration de tout le texte de l'ebook nécessite une permission écrite de l'auteur ;
- le contenu de l'ebook ne doit sous aucune manière que ce soit être modifié. Il doit rester fidèle à sa version d'origine tel que téléchargé ;
- Les références aux marques, aux entreprises et aux universités citées dans cet ebook n'ont en aucune façon que ce soit un but publicitaire, elles sont utilisées exclusivement à des fins académiques et restent entièrement la propriété de leurs détenteurs ;
- Les marques citées dans cet ebook, les logos d'entreprises, sont des marques déposées des entreprises en France, aux états Unis ou partout dans le monde ;
- Les conseils, les tableaux comparatifs, les benchmark de solutions et les prises de position présents dans l'ebook représentent le point de vue personnel de l'auteur à la date de publication. Aucun favoritisme n'a été fait lors des benchmarks et des comparaisons. Etant donné la vitesse avec laquelle évolue le monde de la technologie et du Big Data, beaucoup de ces conseils et tableaux peuvent devenir obsolètes après la publication de cet ebook. Ainsi, bien que l'auteur ait pris tous les soins nécessaires afin de vous aider à travailler dans le Big Data, il ne peut être tenu pour responsable des résultats négatifs qu'auraient causé l'application de ces conseils après la date de publication de l'ouvrage ;
- En raison des changements rapides du marché, le contenu des sites Web fournis peut être modifié ou changé, ou le site Web lui-même peut être indisponible. Donc, après la date de publication de cet ebook, l'auteur ne peut vous donner aucune garantie quant à la disponibilité des sites Internet fournis ;
- Il y'a de l'anglicisme dans cet ebook. C'est un choix personnel de l'auteur, qui veut par-là conserver la fiabilité de la teneur sémantique des mots du jargon ;
- Toute personne qui exécutera un acte non-autorisé à l'égard de cet ebook (recopie de tout l'ebook sans permission écrite de l'auteur, recopie partielle de l'ebook sans mentionner l'auteur, utilisation commerciale) s'expose à des poursuites judiciaires conformément aux dispositions du Copyright en vigueur en France, dans l'Union Européenne et dans le monde.

Copyright 2020 © Juvénal CHOKOGOUE

INTRODUCTION

Vous souhaitez devenir Data Engineer ? Vous souhaitez vous orienter vers les métiers de la Data ? Le moment est approprié pour vous lancer, car c'est la data qui alimente toute les activités de nos société numérique actuelle ; par exemple : la régie publicitaire est de plus en plus basée sur la data, les compteurs intelligents avec Linky d'EDF, l'Intelligence Artificielle, l'agriculture verte, les véhicules hybrides, les objets intelligents, les objets connectés, le smart computing, etc... Bref, **la viabilité de la majorité des modèles économiques de notre époque dépend de l'exploitation intelligente de la donnée.**

En fait, pour gérer les charges de calcul engendrées par le traitement des volumes astronomiques de ces données, un cluster est nécessaire. Le cluster en réalité est juste l'infrastructure technique. Pour qu'il soit opérationnel, il a besoin d'un modèle de calcul. Le modèle de calcul spécifie la façon dont les tâches sont parallélisés dans les nœuds du cluster.

La majorité des modèles de calcul disponibles sur le marché à l'heure actuelle ont l'avantage de combiner les fonctionnalités du batch Processing à la performance d'un cluster pour traiter des problèmes par définition simples à paralléliser.

Le MapReduce, le modèle pionnier dans ce domaine, est le modèle qui est à l'origine d'[Hadoop](#). Malgré sa simplicité, il n'est pas adapté à toutes les problématiques, précisément les problématiques impliquant le traitement interactif et les problématiques impliquant le traitement itératif des données, comme les problématiques de Data Science, apprentissage scientifique, streaming, reporting temps réel, etc.

Spark a été conçu à l'origine pour répondre à ces limites. Il est adapté aussi bien pour le calcul Interactif que pour le calcul itératif. Il affiche des performances qui sont 10 fois plus élevés que Hadoop sur les travaux itératifs (apprentissage statistique).

Avec le temps, grâce à des abstractions incroyablement performantes (comme le RDD, le Dataframe, et le Data set. Nous y reviendrons plus bas) et ses modules diversifiés sur les problématiques les plus récurrentes du Data Engineering, Spark a su s'imposer en tant que LE framework de calcul massivement nécessaire pour valoriser les données !

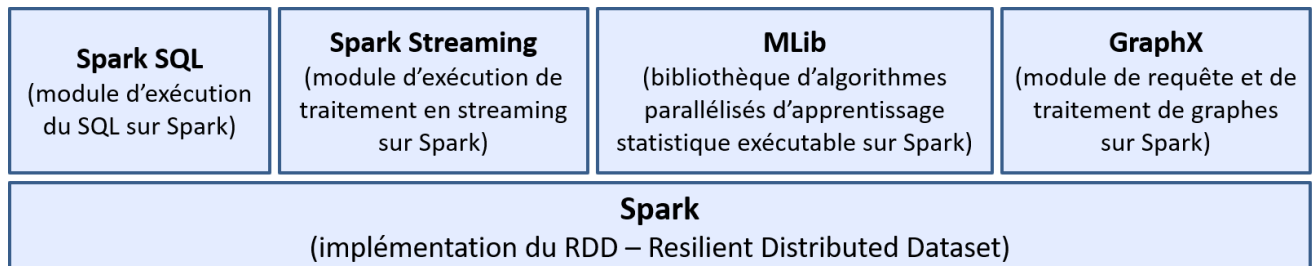


Figure : briques de Spark

Spark est devenu tellement présent dans les problématiques de traitement de données que sa maîtrise est aujourd'hui obligatoire dans quasiment toutes les missions Big Data, spécialement pour les postes/missions de Data Engineer ou Tech Lead. Faites un tour sur les appels d'offre postés sur Indeed, tapez « data engineer », « big data », et vous verrez que pour eux tous, la maîtrise de Spark est OBLIGATOIRE ! Le niveau minimum de maîtrise exigé par les clients n'est pas « débutant », mais c'est le niveau "confirmé". Ainsi, si vous avez vraiment pour projet de travailler dans le Big Data et de devenir Data Engineer, vous devez absolument maîtriser Spark. Ici, je ne parle pas de « connaître » dans le sens d'avoir fait quelques TD ou d'avoir suivi quelques tutoriels, je parle de maîtrise !! Car seule cette dimension vous permettra de véritablement apporter de la valeur dans un projet Big Data.

Aussi, quoique exigeant, la maîtrise de Spark est devenue très lucratif. En effet, selon les estimations de **Glassdoor**, le salaire moyen d'un(e) Data Engineer (H/F) (France) maîtrisant Spark est de **43 771 € / an** au moment de la publication de ce livre numérique. Il s'agit là d'une estimation basée sur 149 salaires postés anonymement sur Glassdoor par des employés occupant le

poste de Data Engineer. Le salaire minimum affiché est de 36 000 €, tandis que la fourchette haute est à 61 000 €.

Selon les estimations d'**Indeed**, le salaire moyen est de **47 044 € / an**. Il s'agit d'une estimation fondée sur 272 salaires envoyés de manière anonyme à Indeed par des employés (Ingénieur Big Data (H/F)) et des utilisateurs, ainsi que sur des offres d'emploi actuelles ou publiées sur Indeed au cours des 36 derniers mois. Il faut noter qu'à la différence des autres jobboard, Indeed indique également les variations de salaires en fonction de la zone géographique. Par exemple, le salaire moyen d'un(e) Ingénieur Big Data (H/F) pour des profils seniors Spark en île-de-France est de **52 917 €** par an. En matière de TJM, on est sur des tarifs variant entre 479 et 580 euros.

La maîtrise de Spark représente donc une opportunité fabuleuse pour tout ceux qui désirent s'orienter dans le marché de la data en tant que Data Engineer.

Dans ce livre, nous allons vous mettre sur les pistes de valorisation de données en Big Data avec Spark. **L'idée est de vous expliquer les bases de l'utilisation de Spark pour traiter des cas d'usage que vous rencontrerez chez vos clients en tant que Data Engineer.**

Maintenant que le décor est planté, entrons dans le vif du sujet !

1- Spark : le moteur Big Data 4-en-1

La base en Big Data c'est le MapReduce. Le MapReduce est un modèle algorithmique dans lequel le traitement des données est découpée en 3 tâches interdépendantes **Map → Shuffle → Reduce** et exécutées en batch sur le cluster. La figure ci-après l'illustre, l'ordre d'exécution qui est imposé au cluster est séquentiel et non-itératif.

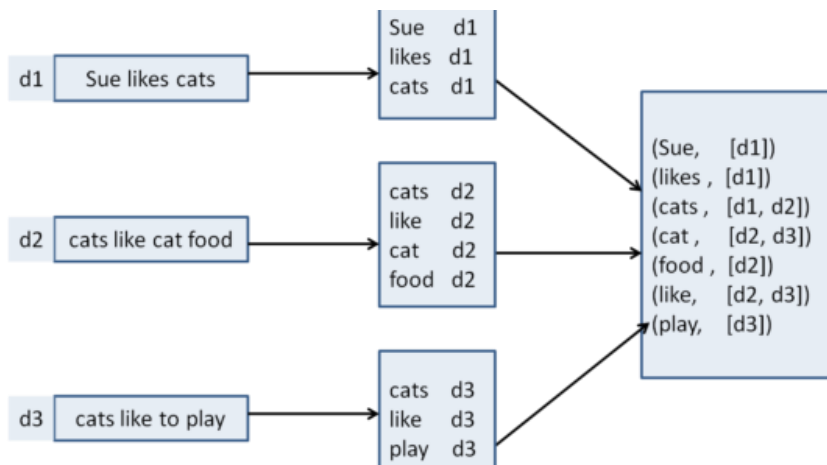


Figure : le MapReduce est modèle orienté batch qui s'exécute de façon séquentiel et direct.

Pendant longtemps (et même encore jusqu'à présent), le MapReduce était utilisé dans la majorité des problématiques du Big Data. Le MapReduce fonctionne très bien, surtout sur les problématiques de Reporting, Business Intelligence, qui sont en général des travaux orientés-batch

Le problème c'est que même si les modèles batch comme le mapreduce permettent d'exploiter au max la caractéristique de « commodité » des cluster, ils ne sont pas adaptés pour certaines applications, notamment celles qui réutilisent les données à travers de multiples opérations telles que la plupart des algorithmes d'apprentissage statistique, itératifs pour la plupart, et les requêtes interactives d'analyse de données.

Spark est une réponse à ces limites. **Spark est un moteur de calcul In-Memory Distribué.** En tant que moteur In-Memory, il excelle sur les tâches itératives et interactives et sa particularité est qu'il excelle sur ces tâches tout en conservant la scalabilité et la tolérance aux pannes du cluster.

Spark est sous licence Apache depuis Février 2014, et au même titre qu'Hadoop, fait partie désormais des technologies de l'écosystème Big Data.

Si Spark a réussi à s'imposer aujourd'hui dans les problématiques du Big Data c'est grâce à ses 4 modules, spécialisés chacun dans une des problématiques particulières du Data Engineering : *Spark SQL, Spark Streaming, Spark MLlib*

et *GraphX*. La figure suivante illustre ces composants.

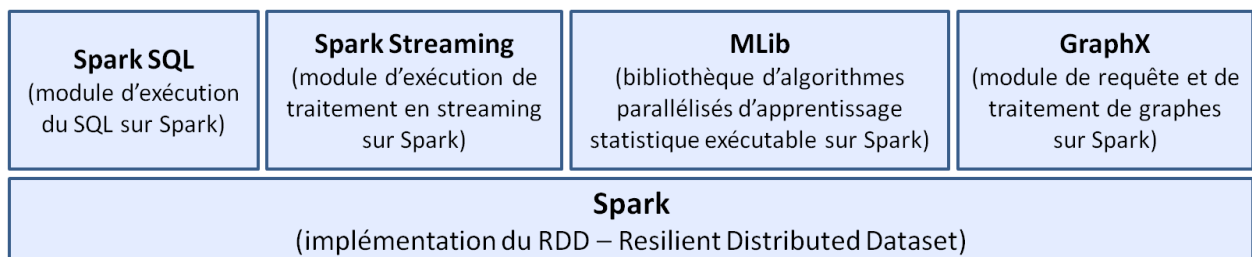


Figure 1 : composants de Spark

- 1- **Spark SQL** est un module qui permet d'écrire des requêtes SQL ou des requêtes HiveQL et de les exécuter sur Spark. Cela permet aux professionnels non-informaticiens utilisant Hive de migrer sur Spark facilement. Ce module fournit une abstraction très intéressante pour la manipulation des données comme nous le verrons plus bas. De plus, il permet d'utiliser Spark comme un client et d'interroger des bases de données distantes hébergées aussi bien dans des bases relationnelles (MySQL, SQL Server, Oracle, Postgre, etc), que non-relationnelles (Cassandra, HBase, ElasticSearch, etc).
- 2- **Spark Streaming** est un module qui permet de développer des applications de données produites au fil de l'eau ou en streaming. Il va permettre de travailler sur des cas d'usage streaming, comme la production d'indicateurs en temps réel, le reporting à faible latence, la détection de fraude, etc.
- 3- **Spark MLib** est une bibliothèque d'algorithmes d'apprentissage statistiques parallélisés qui permet de traiter la majorité des problèmes d'apprentissage statistique de façon interactive sur Spark. Vous pourrez grâce à ce module, effectuer des travaux de recommandation, de clustering, la régression logistique, les applications de ces algorithmes sont nombreuses...
- 4- **Spark GraphX** est un module qui permet d'analyser et d'exécuter des requêtes sur un graphe (par exemple le graphe d'un réseau social). Grâce à ce composant, vous serez capable d'interroger une base de données orientée graphe, ou des cas d'usage manipulant des données en structure de graphe, comme le séquençage du génome humain.

C'est en réalité grâce à ces 4 modules interdépendants que Spark est si

apprécié, et est capable de couvrir au moins 80% des cas d'usage du Big Data. J'aime appeler Spark, **le moteur Big Data In-Memory 4 en 1**.

Ces 4 modules de Spark peuvent être exploités à l'aide de l'un des trois langages de programmation suivant : Scala, Java et Python.

Originellement, Spark a été développée en Scala. Du coup, il bénéficie de ses fonctionnalités, dont 3 particulièrement :

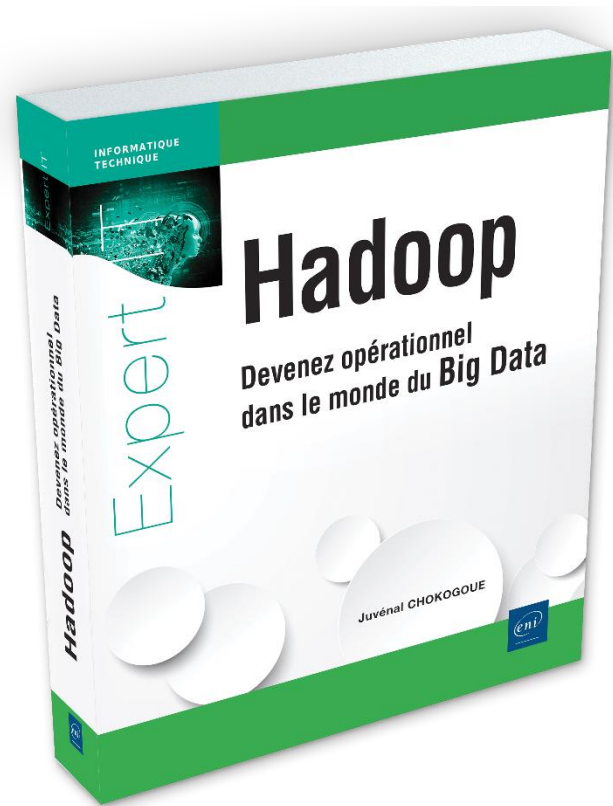
- **l'Immutabilité** : Spark est immuable. Cela lui est très utile lors de l'exécution parallèle des travaux dans le cluster. Nous y reviendrons un peu plus bas.
- **Lazy computations** : les calculs sont "paresseux" par défaut avec Spark. Cela signifie que Spark exécute les expressions uniquement lorsqu'elles sont nécessaires. Cela améliore grandement les performances de l'application. Nous y reviendrons plus bas.
- **les collections d'objets** : la caractéristique clé de Scala ce sont ses structures de données flexibles et puissantes ! On les regroupe sous le vocable « collection ». Spark hérite également des caractéristiques des collections de Scala, notamment via le RDD. Nous allons y revenir plus bas.

Plus haut, nous avons expliqué que le cluster et le MapReduce sont les fondations du Big Data.

***Vous débutez dans le Big Data ?
les notions de cluster,
MapReduce, In-memory ne vous
disent pas grand-chose ?***

Notre ouvrage [« Hadoop – Devenez opérationnel dans le monde du Big Data »](#), le premier livre du projet, vous aidera à développer ces compétences.

Le livre est l'ouvrage N°1 en la matière dans toute la francophonie !!



★★★★★ **Excellent ouvrage !!!**
Par [Bonnet Stephane](#) le 13 août 2017
Format: Broché | **Achat vérifié**

★★★★★ **Excellent**
Par [Aus](#) le 5 février 2018
Format: Broché | **Achat vérifié**

★★★★★ **Un ouvrage informé, intelligent et concret !**
Par [Carole Njoya, Risk Manager Expert ERM](#) le 12 novembre 2017
Format: Broché

2- LES RDD : LA CLE DE VOÛTE DE LA PERFORMANCE DE SPARK

Si Spark est autant plébiscité en Big Data, ce n'est pas seulement à cause de la diversité de ses modules, mais aussi et surtout grâce à ses performances inégalées en matière de calcul massivement parallèle. En effet, d'après des tests menés par l'université de Berkeley, **Spark affiche des performances qui sont 10 fois plus élevés que Hadoop sur les travaux itératifs (apprentissage statistique)**. L'université a mené une expérience dans laquelle elle compara Spark et Hadoop sur la performance d'un job de régression logistique, un algorithme itératif de classification qui essaye de trouver une droite (un hyperplan pour être plus formel) qui sépare au mieux 2 ensembles de données. Pour ce faire, un fichier de 29 Go a été utilisé sur un cluster de 20 machines *EC2 M1.XLARGE*¹ possédant 4 CPU chacune. Les résultats sont là : avec Hadoop, chaque itération prend 127 secondes, tandis qu'avec Spark, la première itération prend 174 secondes, mais les itérations suivantes prennent 6 secondes seulement. Une autre expérience a été menée dans laquelle Spark a été utilisé pour charger 39 Go de fichiers Wikipedia en mémoire à travers un cluster de 15 machines *EC2 M1.XLARGE* et y effectuer des requêtes interactives. La première fois que la requête a été traitée, elle a mis 35 secondes pour être traitée par Spark. Cependant, les requêtes suivantes ont pris entre la demi-seconde et la seconde. Ces expériences montrent que sur les travaux itératifs et les travaux interactifs, Spark est de loin meilleur à Hadoop !

En réalité, si Spark se démarque autant d'Hadoop en matière de performance, c'est parce qu'au lieu de se baser sur un modèle de calcul batch exécuté directement de façon acyclique, comme la majorité de modèles de traitements Big Data, les ingénieurs de Spark ont plutôt choisi de créer une abstraction In-Memory distribué qui tournerait sur un cluster configuré en Shared-Nothing.

¹ *EC2 M1.XLARGE* ce sont des machines d'une configuration particulière offertes dans l'offre Cloud EC2 (Elastic Compute Cloud) d'Amazon : <https://aws.amazon.com/fr/ec2/previous-generation/>

Cette abstraction distribuée s'appelle le **RDD - Resilient distributed dataset** (*jeu de données distribué et résilient*). Le RDD est une "collection" d'éléments partitionnée à travers les nœuds du cluster. Grâce au RDD, Spark parvient à exceller sur les tâches itératives et interactives tout en conservant la scalabilité et la tolérance aux pannes du cluster.

Ainsi, si vous voulez exceller dans vos compétences Spark, la première étape consiste à bien comprendre ce qu'est le RDD, comment il fonctionne, et savoir comment l'utiliser efficacement dans les cas d'usages Big Data. C'est ce que nous allons nous évertuer à faire maintenant.

2.1. Comprendre les RDD

le RDD, **Resilient Distributed Dataset** (*Table Distribuée et Résiliente* ou *Jeu de données résilient et distribué*) est une abstraction distribuée en mémoire qui permet aux développeurs d'effectuer des calculs parallèles en mémoire sur un cluster de façon complètement tolérante aux pannes. Le RDD a été créé pour résoudre le problème que pose les algorithmes itératifs et les calculs interactifs au MapReduce.

Formellement, le RDD est une « collection » (au sens Scala du terme) d'éléments partitionnée à travers les nœuds du cluster et accessible uniquement en lecture-seule. Les ingénieurs de Spark se sont appuyés/inspirés du fonctionnement des collections de Scala (les tableaux, les listes et les tuples) pour concevoir le RDD. On pourrait même dire d'un point de vue pratique que les RDD sont des collections Scala distribuées. Nous y reviendrons un peu plus bas.

Les RDD agissent comme une abstraction de partage de données dans un cluster. Lorsqu'on parle de « distribué » dans la définition de RDD, on fait en réalité référence à « **shared** » (qui a une sémantique ou un sens différent de « *distributed* »), partagé, atomique, car Il utilise la mémoire cache pour persister les données en RAM pour réutilisation, évitant ainsi la réplication des données sur disque, nécessaire dans Hadoop pour garantir la disponibilité du cluster.

C'est grâce à ce mécanisme qu'il est capable de fournir de la haute disponibilité et la tolérance aux pannes au cluster.

Etant donné qu'un RDD est une abstraction, elle n'a pas d'existence réelle, dès lors elle doit être explicitement créée ou instanciée à travers des opérations déterministes sur des fichiers de données existants ou sur d'autres instances de RDD. ces opérations d'instanciation de RDD sont appelées des "**transformations**". Nous y reviendrons plus bas.

Comme nous l'avons mentionné plus haut, il est important de toujours garder à l'esprit que Spark a été développé en Scala et que les RDD, en tant que collection, ont simplement hérité des caractéristiques des collections scala (tableau, liste, tuple), à savoir :

- **Lazy computations** : les calculs exécutés sur les RDD sont "paresseux". Cela signifie que Spark exécute les expressions uniquement lorsqu'elles sont nécessaires. Techniquement, c'est lorsqu'une action est déclenchée sur le RDD que celui-ci est exécuté. Cela améliore grandement les performances de l'application, puisque ses différents scripts n'ont pas à être exécuté s'ils ne sont pas sollicités directement pour un résultat final, consommable de suite par l'utilisateur. Vous imaginez le gain de temps gagné par rapport à une application classique ou par rapport aux autres technologies de l'écosystème Hadoop, qui exécute les scripts de code ligne-par-ligne jusqu'à la fin !

- **Immutable** : les RDD sont immutables. Cela signifie qu'ils sont accessibles en lecture-seule uniquement. On ne peut donc pas modifier un RDD. Cette caractéristique est très utile lors de la gestion d'accès concurrente aux données, spécialement dans un contexte de valorisation de données à large échelle... C'est avec la pratique que vous comprendrez véritablement l'intérêt de cette caractéristique.

- **In-memory** : les RDD sont exécutés en mémoire. Ils peuvent également être persistés en cache pour plus de rapidité. Les développeurs de Spark ont prévu

la possibilité de choisir où persister les RDD (soit sur disque dur, soit en cache, soit en mémoire et sur disque) grâce à la propriété *Storage.LEVEL*.

2.2. Utiliser les RDD de Spark

Spark expose ou met à disposition des utilisateurs les RDD à travers une API développée en Scala (langage de base), en Java, et en Python. Les jeux de données dans les RDD sont représentées comme des objets (instances de classe) et les transformations sont invoquées en utilisant les méthodes de ces objets. L'aspect fonctionnel de Scala se prête très bien à ce style d'opérations. C'est pourquoi, pour tout développement applicatif, je recommande personnellement l'utilisation de Spark avec Scala ou Java (Scala est moins verbeux que Java, plus expressif et donc plus simple à maintenir). Par contre, si vous êtes sur des cas d'usage de Data Science, PySpark, l'API de Spark en Python est plus appropriée.

pour utiliser Spark, vous écrivez un programme pilote (un *driver*) qui implémente le flux de contrôle de haut-niveau de votre application et lance des opérations variées en parallèle. Spark fournit 2 abstractions principales pour la programmation parallèle en langage de programmation : les transformations de RDD et les opérations parallèles sur ces RDD.

En fait, utiliser les RDD revient à effectuer des ***transformations***, qui sont des opérations lazy qui créent un nouveau RDD sur la base soit des fichiers de données localisés ou non sur le HDFS, soit alors des instances de RDD existants, et finit par l'utilisation des ***actions***, qui sont quant à elles des fonctions qui retournent une valeur à l'application. Un exemple d'action inclut la méthode "*count()*" qui retourne le nombre d'éléments dans le jeu de données, "*collect()*" qui retourne les éléments eux-mêmes, "*for each :*" qui boucle les données dans une fonction fournie par l'utilisateur ou encore "*persist()*" qui persiste un RDD en mémoire. **Le cœur de la programmation des RDD se situe vraiment sur l'utilisation des transformations et des actions.** C'est vraiment la base pour monter en compétence en Spark ! Le tableau suivant

fournit un ensemble non-exhaustif des transformations et actions disponibles en programmation Spark.

Tableau : liste des transformations courantes des RDD

Transformations	Signification
<i>Map()</i>	Crée un nouveau RDD avec les conditions spécifiées dans la fonction anonyme
<i>Filter()</i>	Crée un nouveau RDD contenant les données répondant aux conditions de filtre
<i>Flatmap()</i>	Crée un nouveau RDD au même titre que <i>map()</i> , à la seule différence que le nouveau RDD est aplati.
<i>Sample()</i>	Crée un nouveau RDD constitué d'un échantillon de données du RDD source.
<i>Groupbykey()</i>	Effectue une opération <i>GroupBy()</i>
<i>Reducebykey()</i>	Effectue une opération <i>Reduce()</i>
<i>Union()</i>	Union de deux ou plusieurs RDD
<i>Join()</i>	Jointure de deux ou plusieurs RDD
<i>Sort()</i>	Tri du RDD

Tableau : liste des actions courantes des RDD

Actions	Signification
<i>Count()</i>	<i>Compte le nombre d'éléments présents dans le RDD</i>
<i>Collect()</i>	<i>Transfère les données du RDD dans le nœud principal (ou driver) du cluster Spark</i>
<i>Reduce()</i>	<i>Effectue une agrégation entre les éléments du RDD. Il combine les données en utilisant une fonction associative qui produit un résultat au programme pilote.</i>
<i>Save()</i>	<i>Persiste les données du RDD sur le disque dur</i>
<i>Persist()</i>	<i>Persiste les données du RDD (provenant du cache)</i>
<i>Sum()</i>	<i>Effectue une somme des éléments du RDD</i>
<i>Foreach()</i>	<i>Permet d'effectuer des actions spécifiques pour chaque élément du RDD</i>

Concrètement, vous pouvez construire les RDD à partir de l'une des 4 voies suivantes :

- à partir d'un fichier dans le système de fichier distribué, comme le HDFS,
- en parallélisant une collection Scala dans le programme pilote dont les partitions seront réparties dans le cluster,
- en transformant un RDD existant.
- en sérialisant un RDD existant. Par défaut, les RDD sont chargés en mémoire. Les partitions du RDD sont sérialisées/persistées sous-demande lorsqu'ils sont utilisés dans une opération parallèle (par exemple : en passant un bloc de fichier à une fonction map), et sont supprimées de la mémoire après utilisation.

A ce stade, vous avez toutes les informations nécessaires pour commencer à pratique Spark ! C'est ce que nous allons faire dans la session suivante.

3- PRATIQUE DE SPARK !

Nous allons maintenant vous aider à faire vos premiers pas sur Spark. La connaissance des RDD est suffisante pour commencer à écrire ses premières applications. La première chose consiste à installer Spark sur votre machine.

3.1. Installer Spark sur son PC

L'une des plus grande difficulté dans l'apprentissage de Spark concerne la mise en place d'un environnement. En effet, Spark s'exécute sur un cluster et il est quasiment inenvisageable de trouver un cluster pour des fins personnelles.

Une solution à ce problème serait d'utiliser une machine virtuelle contenant une distribution Hadoop, telle que [la machine virtuelle offerte par Cloudera](#). Le problème est que ces machines virtuelles sont très lourdes et nécessitent souvent au minimum 16 Go de RAM pour correctement s'exécuter sur le PC. Les PC classiques sont composés de 8 Go de SDRAM (ou 12 Go pour les plus équipés comme le mien). Au-delà de 8 Go de mémoire en SSD de disque, le prix du PC commence à dépasser les 1 200 euros.

Une seconde alternative est envisageable, le Cloud. Ici, il s'agit d'obtenir l'accès à une distribution Hadoop dans le Cloud, telle que [Amazon EMR](#), [Google GCP](#), ou encore [Microsoft Azure HDInsight](#). Quoique cela ne soit pas coûteux à l'heure, sur l'ensemble du temps dont vous aurez besoin personnellement pour maîtriser complètement Spark, cela vous coûtera une petite fortune ; Nous estimons à 3 mois, le temps minimum nécessaire pour être opérationnel sur Spark. Vous n'obtiendrez le niveau d'expertise qu'en travaillant sur des projets réels en entreprise.

Il y'a une autre alternative, qui est celle que nous avons retenue, qui ne vous coûte rien financièrement, ni n'exige d'investissement sur un PC super-puissant. Il s'agit de l'installation en standalone de Spark sur votre PC windows/mac OS/Linux classique. Bien que cette alternative soit la plus simple et la moins

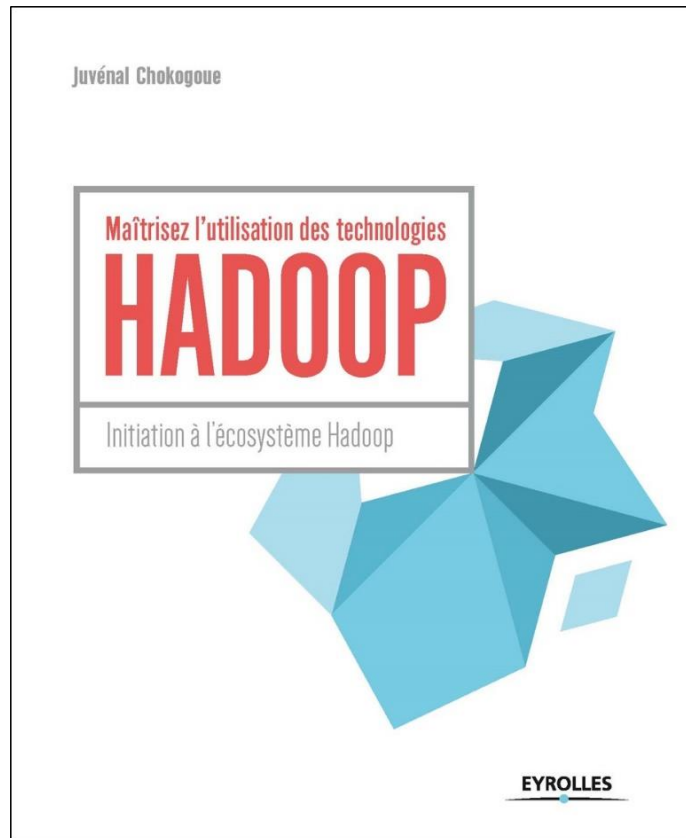
coûteuse des trois options, elle nécessite quand même de la prudence et une certaine expertise pour éviter certains écueils qui seront fatales à vos déploiement d'applications plus tard.

Comme vous l'imaginez peut-être, l'installation de Spark sur votre PC n'est pas quelque chose que nous pouvons faire efficacement par écrit. Par conséquent, nous vous offrons en complément de ce livre, **une session vidéo de 62 minutes** dans laquelle nous allons ensemble pas-à-pas vous montrer comment installer Spark sur votre PC pour commencer à développer des applications Big Data.

La sessions est disponible sur le lien suivant : [Installer Spark sur son PC classique](#)

Attention !!! Il est important pour la réussite de l'installation de Spark dans votre PC que vous répétiez les actions telles que vous nous voyez faire dans la session de cours.

En réalité, la plus grosse difficulté dans l'apprentissage de Spark n'est pas son installation. La plus grosse difficulté repose sur le fait que Spark évolue dans un écosystème de technologies complexes et peu connues du grand public, qui fait que si on souhaite déployer les applications Spark qu'on apprend à développer, il faut également maîtriser tout un panel de technologies qui requiert chacune un savoir-faire particulier. Il ne suffit donc pas de maîtriser uniquement les API de Spark, **vous devez également maîtriser l'écosystème de technologies autour de Spark pour être en même de l'utiliser à un niveau professionnel.** L'ensemble de cet écosystème technologique s'appelle **l'écosystème Hadoop**. Dans le 2nd ouvrage du projet, nous vous aidons à monter en compétence **sur les 18 technologies clé de l'écosystème Hadoop**, qui tourne auprès de Spark. Cliquez sur l'image suivante pour obtenir une copie de l'ouvrage.




 garantie

★★★★★ Magnifique Bouquin !

21 juillet 2018

Format: Broché

 tourist again

★★★★★ Précision, maîtrise et pédagogie (plus besoin d'acheter des ouvrages en anglais :-)

14 août 2018

Format: Broché

 Franck Perle

★★★★★ Excellent

10 juillet 2018

Format: Broché | Achat vérifié



3.2. Créer une session Spark

A ce stade, vous avez installé Spark sur votre PC sans que cela vous coûte un investissement financier supplémentaire. Une fois que vous avez installé Spark, la deuxième étape pour démarrer son apprentissage consiste à créer une session.

Pourquoi ?

Parce que peu importe ce que vous souhaitez développer, le point d'entrée à toute application Spark c'est la **session Spark**. Nous en avons longuement parlé dans notre ouvrage "[Maîtrisez l'utilisation des technologies Hadoop](#)".

La session Spark permet d'instancier Spark sur le cluster, et de définir les ressources qu'elle utilisera pour s'exécuter dans le cluster.

En théorie, développer une session Spark est simple et revient à une seule chose :

```
import org.apache.spark.sql.Session
val spark = Session.builder.appName("application
simple").getOrCreate()
```

Mais dans un contexte professionnel, développer la session peut s'avérer rapidement très compliqué. Celle-ci se crée à l'aide de la classe `Session`. Pour construire une session Spark, il suffit d'utiliser la méthode "builder" de la classe `Session`. Un exemple basique :

```
val spark = Session.builder.appName("application
simple").getOrCreate()
```

Le `Session` builder contient des méthodes qui permettent de spécifier la configuration de la session Spark. Ces méthodes sont :

- `appName` : pour définir le nom de l'application qui sera affiché dans le Spark Web UI
- `config` : pour indiquer une configuration spécifique
- `enableHiveSupport` : pour permettre la connexion au métastore de Hive

- *master* : pour indiquer l'URL du nœud maître du cluster sur lequel tourne Spark (eg : local[*] pour exécution en local)
- *getOrCreate()* : pour créer la session Spark sur la base des paramètres, ou appeler une session Spark existante.

Par expérience, les paramètres à renseigner lors d'une session Spark sont les suivantes :

```
.config("spark.serializer",  
"org.apache.spark.serializer.KryoSerializer")  
  
.config("spark.sql.crossJoin.enabled", "true")
```

eg de session Spark complète :

```
val sparkSession = SparkSession.builder  
    .appName(applicationName)  
    .config("spark.serializer",  
"org.apache.spark.serializer.KryoSerializer")  
    .config("spark.sql.crossJoin.enabled", "true")  
    .enableHiveSupport()  
    .getOrCreate()
```

Comme je vous l'ai expliqué plus haut, lorsqu'on est dans une configuration professionnelle, développer une session Spark peut rapidement devenir très complexe. Encore une fois, nous ne pouvons pas aborder tous les cas de configuration possible par écrit. C'est pourquoi nous mettons à votre disposition, **une seconde session de cours vidéo de 33 minutes**, dans laquelle nous vous expliquons pas-à-pas comment développer une session Spark ainsi que l'ensemble des paramètres qui entrent en compte dans son développement.

Cliquez sur le lien suivant pour suivre le tutoriel et **répétez les actions telles que vous nous voyez faire dans la session de cours** : [Développer une session Spark efficace](#)

3.3. Savoir manipuler les RDD

Souvenez-vous, plus haut dans la section 2, nous avons expliqué que le RDD est la clé de voûte de la performance de Spark. Nous avons également expliqué que vous pouvez le créer à partir de 4 sources :

- à partir d'un fichier dans le système de fichier distribué, comme le HDFS,
- en parallélisant une collection Scala dans le programme pilote dont les partitions seront réparties dans le cluster,
- en transformant un RDD existant.
- en sérialisant un RDD existant. Par défaut, les RDD sont chargés en mémoire. Les partitions du RDD sont sérialisées/persistées sous-demande lorsqu'ils sont utilisés dans une opération parallèle (par exemple : en passant un bloc de fichier à une fonction map), et sont supprimées de la mémoire après utilisation.

Une fois que vous avez installé Spark, il vous faut un EDI – environnement de développement intégré pour faciliter l'écriture de vos applications. Nous vous recommandons en EDI **IntelliJ** ou **Eclipse**. Ces 2 EDI sont particulièrement bien adaptés respectivement pour Scala Spark (IntelliJ), et Spark Java (Eclipse).

Une fois que vous avez l'EDI, il faut développer une session Spark. Une fois que la session Spark est sur pied, vous pouvez démarrer la manipulation des données via les RDD.

Voici quelques exemples en [Scala sur IntelliJ](#) :

```
def main(args: Array[String]): Unit = {
    val spark: SparkSession =
SparkSession.builder().master("local[1]")
    .appName("Session Spar1")
    .getOrCreate()

    val rdd_test : RDD[Int] =
spark.sparkContext.parallelize(List(1,2,3,4,5))
    val rddCollect: Array[Int] = rdd.collect()
```

```
println("Action: premier element: "+rdd.first())
rddCollect.foreach(println)
}
```

```
val rdd=spark.sparkContext.parallelize(Seq(("Math", 500),
    ("Chimie", 400), ("Java", 300)))
rdd.foreach(println)
```

- à partir d'une source de données grâce au RDD.textfile()

```
println("read all text files from a directory to single RDD")
val rdd = spark.sparkContext.textFile("C:/tmp/files/*")
rdd.foreach(f=>{
    println(f)
})
```

Dans notre formation « [Maîtrisez Spark pour le Big Data avec Scala](#) », vous ressortirez expert dans la manipulation des RDD.

3.4. Les Data frames : la clé de voûte de Spark SQL

Une autre abstraction qui rend Spark très attractif pour le développement des applications Big Data en dehors des RDD c'est le **Data frame**. Les compétences de base en Spark résume uniquement en ces 2 abstractions : *RDD et Data Frame*. Toute application que vous bâtirez en Spark aura toujours pour point d'assise l'une de ces 2 abstractions. C'est pourquoi nous ne pouvons pas quitter ce livre sans vous aider à comprendre les Data Frames.

Théoriquement, un data frame est un "**dataset**" organisé en colonnes nommées. En réalité, C'est l'abstraction d'un RDD, sous forme de table équivalente à la table dans une base de données relationnelle ou un data frame dans Python/R.

Techniquement, un data frame est une collection distribuée de données assise sur l'intersection du RDD et du SQL. En d'autres termes, c'est une abstraction de RDD qui permet de bénéficier à la fois des fonctionnalités des RDD et du moteur d'exécution SQL de Spark. Vous vous souvenez en début de la brique

Spark SQL, que nous avons annoncé au début ? En fait, le Data Frame est l'abstraction de cette brique.

Les data frame peuvent être construits à partir d'une large variété de sources de données, telles que : les fichiers de données structurés (csv, txt, dat, etc.), les fichiers non-structurés (json, xml, parquet, orc, etc..), les tables Hive, les tables de bases de données relationnelles (MySQL, PostgreSQL, etc...), les SGBD NoSQL (HBase, Cassandra, Elasticsearch, etc.), et les RDD existants.

Création d'un data frame

la création d'un data frame se fait principalement à partir de la méthode read() d'une session spark. Vous pouvez peupler un dataframe à partir de plusieurs sources de données, telles que les csv, les rdd, les BD externes, le json, le xml, etc... La spécification de chaque source se trouve ici :

<http://spark.apache.org/docs/latest/sql-data-sources.html>

Voici quelques exemples :

```
val df_csv = spark.read.format("csv")
  .option("sep", ";")
  .option("inferSchema", "true")
  .option("header", "true")
  .load("chemin/fichier.csv")
```

```
val df_csv = spark.read.format("csv")
  .option("sep", ";") ou bien .option("delimiter", ";")
  .option("inferSchema", "true")
  .option("header", "true")
  .csv("chemin/fichier.csv")
```

La manipulation des data frame repose sur les mêmes principes que ceux des RDD : vous utiliser des transformations, qui sont exécutées lors des actions. Nous allons maintenant vous expliquer comment sont exécutés les applications dans un cluster Spark. C'est très important que vous le compreniez pour savoir écrire du code performant.

4. EXECUTION DES RDD DANS UN CLUSTER

Pour utiliser Spark, vous écrivez ce que Berkeley appelle un "**Spark driver**" (*pilote Spark*), qui est un peu l'équivalent d'un "*Job MapReduce*" en Hadoop. Ce pilote Spark se connecte aux processus Workers qui tournent au niveau des nœuds de données du cluster, ensuite le pilote définit une ou plusieurs RDD sur lesquelles il déclenche une ou plusieurs actions. La figure suivante illustre l'exécution d'un programme dans un cluster Spark.

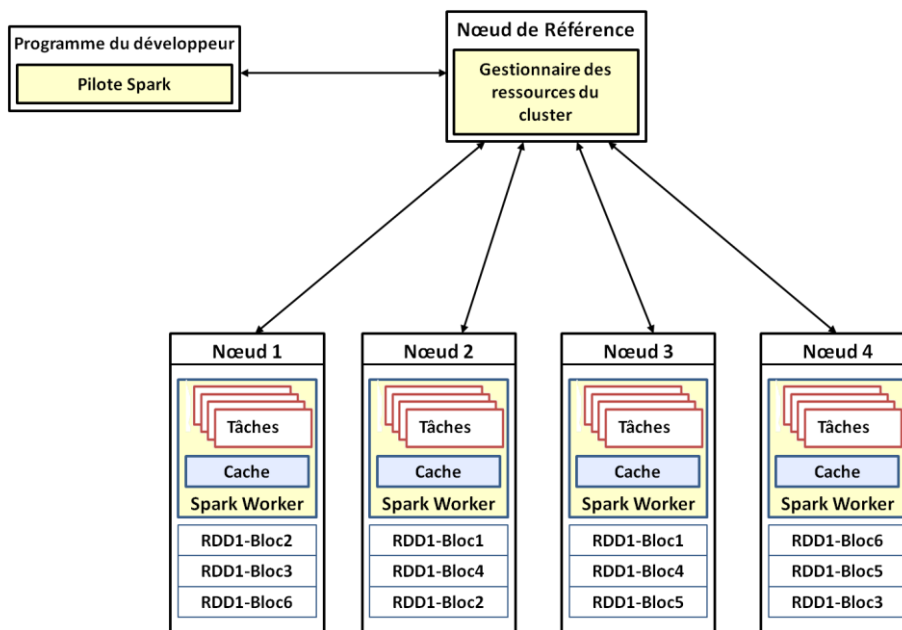


Figure : exécution d'une application spark dans un cluster.

A la différence d'un cluster Hadoop, les processus Workers dans Spark sont des processus d'une durée de vie qui ne s'achève pas à la fin de l'exécution du programme Pilote, ils sont continuellement actifs, c'est pourquoi ils peuvent persister les partitions de RDD en mémoire.

En réalité, c'est la **persistance en mémoire cache des partitions du RDD qui est la clé de la performance et de l'efficacité de Spark**. En effet, les Workers utilisent la mémoire cache pour persister les données dans les partitions de RDD pour réutilisation, évitant ainsi la réplication des données sur disque, nécessaire dans Hadoop pour garantir la tolérance aux pannes du cluster.

A chaque partition du RDD distribué dans les nœuds du cluster, Spark mémorise suffisamment d'informations sur elle de manière à ce que si un nœud venait à tomber en panne, sa partition serait simplement reconstruite à l'aide de ces informations. Ces informations s'appellent le ***lignage*** ou ***lignée du RDD*** (*RDD lineage*).

Le lignage de RDD

Gardez toujours à l'esprit qu'en tant que collection d'objets partitionnés à travers les machines du cluster, les RDD sont ***résilients***. Cela signifie que les partitions d'un RDD peuvent être reconstruites en cas de perte ou de panne sans que ses éléments/objets ne soient persistés sur un disque dur. Pour comprendre cela, il faut déjà comprendre comment la majorité des systèmes de traitement de données à large échelle font pour être tolérant aux pannes.

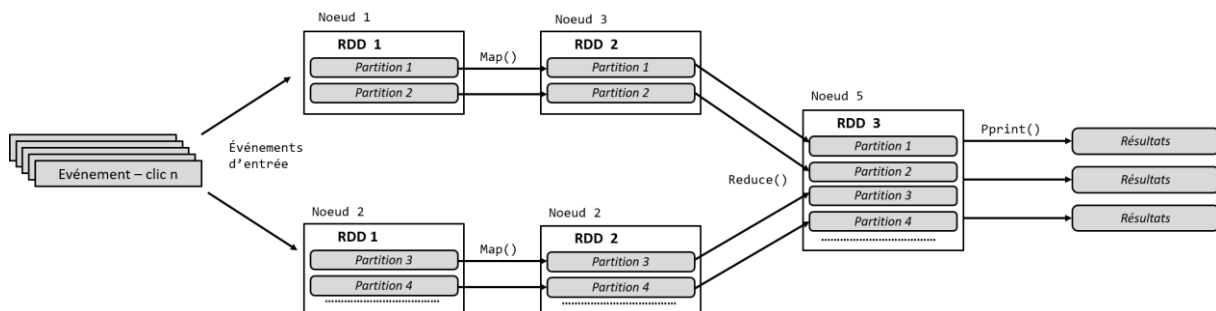
Pour être tolérant aux pannes, la majorité des systèmes Big Data passe par la réplication des données dans le disque dur des autres machines du cluster. La reprise est donc effectuée simplement par restauration ultérieure de ces données. Cette approche quoique pratique, crée un problème sérieux : *assurer la tolérance d'un nœud demande un nœud supplémentaire dans le cluster, en d'autres termes, la réplication demande un doublement ou une multiplication des coûts d'infrastructure du cluster.*

Spark lui passe par un autre mécanisme pour gérer les problèmes de pannes dans le cluster lors du traitement, **la restauration parallèle**. Spark enregistre périodiquement des points de contrôle (***checkpoints***) de l'état des RDD en les répliquant de façon asynchrone dans d'autres nœuds du cluster Spark. Lorsqu'il y a panne de nœud, le système détecte les RDD manquants et relance leur lignage à partir du dernier état enregistré. Comme le RDD est une abstraction nativement distribuée, la réexécution du lignage se fait en parallèle sur tous les nœuds dont les RDD sont manquants, ce qui est plus rapide que la restauration classique qui a lieu par transfert de l'état du nœud de sauvegarde au nœud de calcul.

Un mécanisme qui est capable de restaurer automatiquement les données perdues en cas d'échec ou de panne de système sans réplication est qualifié d'**auto-résilient**.

Les RDD sont auto-résilients, c'est-à-dire qu'ils peuvent restaurer automatiquement les données perdues en cas d'échec ou panne de système sans les répliquer. La restauration se fait par réexécution parallèle du lignage du RDD, c'est-à-dire par réexécution de l'ensemble des opérations qui ont permis de construire le RDD ;

Formellement, le lignage de RDD fait référence à *l'ensemble des opérations qui ont permis de construire le RDD*. Il permet de restaurer le RDD en cas de panne. Le lignage de RDD est similaire à la restauration d'une base de données relationnelle qui consiste à rejouer non pas les données, mais les commandes SQL du log qui ont permis d'obtenir ces données.



Les RDD réussissent la restauration automatique en traçant l'ensemble des opérations nécessaires pour obtenir les données (son lignage), un peu comme le log (journal de transactions) d'un SGBDR. Le log restaure la base de données non pas en stockant toutes les données échangées dans les transactions, mais en stockant les commandes SQL qui permettent d'obtenir ces données. Les opérations du RDD sont tracées dans son lignage sous forme d'un graphe de dépendance similaire à celui du MapReduce.

Combiné à sa capacité à stocker les données en mémoire, le RDD est l'élément qui permet à Spark d'exécuter des traitements en mémoire avec des latences très basses.

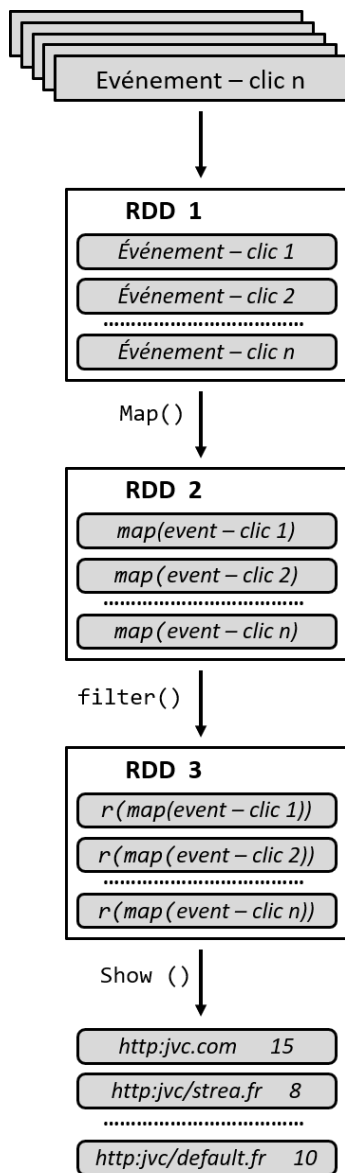


Figure : illustration du lignage d'un traitement MapReduce classique

Attention !!! le lignage n'enregistre pas les données, mais le flux d'opérations qui permettent d'obtenir les données. C'est vrai que nous l'avons déjà dit plus haut, mais c'est important à garder à l'esprit car les autres systèmes enregistrent les états sous forme de données, et c'est ce qui fait que si le nœud contenant la base de données de sauvegarde tombe en panne, le système ne pourra pas restaurer l'état. Le lignage du RDD ne stocke pas de données, ce qui fait que même lorsque le nœud contenant la réplique de l'état du RDD tombe en panne, Spark est toujours en capacité de le restaurer. **Le lignage est très**

important, c'est la clé de voûte de la tolérance aux pannes de Spark.

CONCLUSION

Spark est un système fantastique pour le traitement de données ! Cela à tous égards par rapport à Hadoop ou tout autre système de traitement Big Data. Là où les systèmes classiques de Big Data s'appuient sur un modèle batch acyclique direct (à l'exemple du [MapReduce](#)), qui ne sont pas appropriés pour les calculs itératifs comme la majorité des algorithmes de data science ou de machine learning/deep learning (réseau de neurones, clustering, k-means, regression logistique, etc...), Spark s'appuie sur une abstraction particulière appelée le RDD. Le RDD est une collection d'éléments partitionnés et distribués dans le cluster. Là où la tolérance aux pannes dans les systèmes classiques est obtenue par réplication des données dans le cluster, en Spark celle-ci est obtenue par en traçage de l'ensemble des opérations nécessaires pour obtenir les données présentes dans le RDD. C'est pourquoi ceux-ci sont qualifiés d'auto-**résilients** et sont la fondation de la performance du framework Apache Spark.

En Big Data, la maîtrise de Spark est obligatoire dans la plupart des offres d'emploi et est très lucratif. Comme nous l'avons souligné au début de ce livre, selon de nombreuses études, le salaire moyen d'un(e) Ingénieur Big Data (H/F) pour des profils seniors Spark en île-de-France est de **52 917 €** par an. En matière de TJM, on est sur des tarifs variant entre **479 €** et **580 €**.

Nous avons écrit ce livre pour vous mettre sur les rails de l'apprentissage de Spark et comme vous avez pu le constater vous-même, la maîtrise de Spark demande bien plus qu'1 livre ou 2 ou même 4 livres techniques pointus. Vous avez besoin de formation qualitative qui vous aide à progresser pas-à-pas en fonction de votre emploi de temps.

Nous mettons à votre disposition, une formation pointue qui vous permettra de devenir un spécialiste dans le développement d'applications Spark. La formation est très qualitative et vous aidera à Vous trouverez les détails de cette formation sur le lien suivant ou en cliquant sur l'image suivante : [Maîtrisez Spark pour le Big Data avec Scala.](#)



QUIZZ DE VALIDATION DE COMPETENCES

Vous voulez challenger votre niveau de compétences sur Spark ? Passez le quizz pour consolider les acquis que vous avez obtenu de la lecture de ce livre.

Question : qu'est-ce que l'In-Memory Processing ou traitement en mémoire ?

.....
.....
.....
.....

Question : selon vous, quelle est le meilleur mode de traitement entre le Batch Processing et l'In-Memory Processing ? Justifiez votre réponse

.....
.....
.....
.....

Question : expliquez les deux approches qui peuvent être utilisées pour faire du traitement parallèle In-Memory sur un cluster

1.....
.....
.....
.....
2.....
.....
.....

Question : le Spark est un modèle algorithmique

- Vrai
- Faux

Question : quelle différence faites-vous entre Spark et Hadoop ?

.....
.....
.....
.....
.....

Question : Spark est une version modifiée d'Hadoop

- Vrai
- Faux

Question : qu'est-ce que le RDD et quel est son rôle en Spark ?

.....
.....
.....
.....
.....

Question : pourquoi dit-on que le RDD est résilient ?

.....
.....
.....
.....

Question : pourquoi dit-on que le RDD est distribué ?

.....
.....
.....

Question : le RDD est un modèle de calcul au même titre que le MapReduce

- Vrai
- Faux

Question : Spark est une implémentation du RDD

- Vrai
- Faux

Question : qu'entend-on par « lignage » du RDD (lineage) ?

.....
.....
.....

Question : en quoi consiste la programmation Spark ?

.....
.....
.....

Question : quel mécanisme permet à Spark de rendre un cluster tolérant aux pannes ?

- La réplication des RDD sur les nœuds du cluster
- La persistance des RDD sur le disque dur du nœud
- La persistance des partitions de RDD en mémoire cache

Question : Spark peut tourner sur un cluster Hadoop

- Vrai
- Faux

Question : expliquez brièvement l'exécution de Spark sur un cluster Hadoop

.....

.....

.....

.....

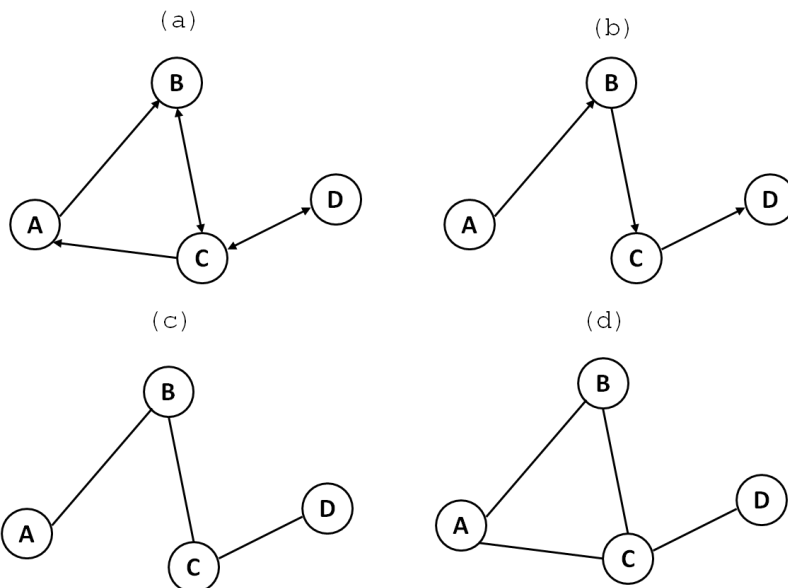
Question : qu'est-ce qu'un graphe acyclique direct ?

.....

.....

.....

Question : soit les 4 graphes suivants :



Répondez aux questions suivantes (indiquez la lettre à chaque fois) :

1-sélectionnez le(s) graphe(s)

direct(s).....

2-sélectionnez le(s) graphe(s)

indirect(s).....

3-sélectionnez le(s) graphe(s)

cyclique(s).....

4-sélectionnez le(s) graphe(s)

acyclique(s).....

5-sélectionnez le(s) graphe(s) cyclique(s)

direct(s).....

6-sélectionnez le(s) graphe(s) cyclique(s)
indirect(s).....

7-sélectionnez le(s) graphe(s) acyclique(s)
direct(s).....

8-sélectionnez le(s) graphe(s) acyclique(s)
indirect(s).....

Question : le MapReduce est un graphe à combien de vertices ?
.....
.....
.....
.....

Question : pourquoi dit-on que le MapReduce est un graphe acyclique direct ?
.....
.....
.....
.....

Question : à quelle problématique correspond Spark ? Cochez la(es) bonne(s) réponse(s)

- L'analyse interactive
- Les jobs itératifs
- Le Streaming

Question de réflexion : Que se passerait-il si on utilisait le modèle MapReduce sur un cluster en Shared-Memory ?
.....
.....
.....
.....

Vous avez fini ? Envoyez-nous vos réponses à contact@data-transitionnumerique.com et nous vous dirons quel est votre niveau sur Spark et les points sur lesquels vous devez encore travailler.

Votre avis compte !

Cher lecteur, encore une fois, merci de vous être procuré cet ebook « *Apprendre le Big Data par la pratique* ». Nous espérons qu'il a répondu à vos exigences et qu'il a rempli ses 3 promesses à votre endroit à savoir :

- vous rendre conscient de l'opportunité financier que représente Spark dans le marché du Big Data
- vous aider à démarrer de bon pied votre montée en compétence sur Spark
- et vous aider à développer les compétences de base indispensables pour utiliser Spark dans les projets réels en entreprise.

Maintenant que vous avez fini sa lecture, n'hésitez pas à en parler autour de vous, ou de recommander le site <http://www.data-transitionnumerique.com/> pour ceux qui souhaitent réorienter leur carrière dans le Big Data.

N'hésitez pas également à entrer en contact avec nous, nous voulons être connectés avec vous, nous serons ravis de connaître ce que vous avez apprécié ou pas de l'ebook, de répondre à vos questions et de vous aider à progresser dans l'acquisition de vos nouvelles compétences. Pour entrer en contact avec nous, vous pouvez :

- Nous écrire directement sur l'adresse : contact@data-transitionnumerique.com

Vous pouvez également nous contacter sur les réseaux sociaux via :

- Facebook: <https://www.facebook.com/transitionnumerique>
- Twitter auteur : https://twitter.com/Juvenal_JVC
- LinkedIn auteur : <https://fr.linkedin.com/in/juvenal-chokogoue>

N'hésitez pas à utiliser ces supports pour tous vos besoins, nous serons ravis d'entrer en contact avec vous et de répondre à toutes vos questions.

Merci encore

Juvénal CHOKOGOUE

Big Data & Streaming : Traitement streaming et temps réel des données en Big Data

Votre livre pour apprendre à construire un data lake et valoriser les données massives
générées en streaming ou en temps réel

Commandez en exclusivité votre copie maintenant
et **Obtenez en exclusivité ces bonus :**



- **Frais de livraison offerts**

Les éditions Juvénal & Associés vous offrent gracieusement les frais de port. Nous supportons à notre charge les frais de livraison de l'ouvrage



- **Autographe de l'auteur**

Vous recevez une copie paraphée de l'autographe de l'auteur.



- **Version numérique offerte**

Une version électronique pdf de l'ouvrage d'une valeur de 35,99 euros vous est offerte en plus de l'imprimé.



Parmi nos clients



MINISTÈRE DE LA DÉFENSE

