ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

SOCIETE GENERALE | BUILDING TEAM SPIRIT TOGETHER

Rapport stage

Cycle : IT3

Utilisation des technologies Big Data pour des analyses de données de risques de marché

Auteur : Amina AIACHE

Le 27 Septembre 2016

Non confidentiel

ECOLE NATIONALE DES SCIENCES GEOGRAPHIQUES
6 et 8 avenue Blaise Pascal - Cité Descartes - Champs sur Marne - 77455 MARNE-LA-VALLEE CEDEX 2
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Jury

Président du jury

Didier RICHARD

Commanditaire :

Olivier CATHALA,

Société Générale (SG)

Tour Société Générale

17, Cours Valmy

92800, PUTEAUX

Encadrement de stage :

An TIAN, Société Générale, maître de stage

Emmanuel BARDIER, ENSG/IGN, référent ENSG

Responsable pédagogique du cycle :

Didier RICHARD , cycle IT3/TSI

Stage du 25/04/2016 au 21/10/2016

Diffusion Web : □ Internet □ Intranet ENSG

Situation du document :

rapport de stage présenté en fin de 3ème année du cycle des ingénieurs

Nombre de pages : 51 dont 0 d'annexes

Système hote : Word

MODIFICATIONS

EDITION	REVISION	DATE	PAGES MODIFIEES
1	0	02/09/2016	Création
1	1	16/09/2016	7-18-27-29

« Shoot for the Moon. Even if you miss, you will land among the stars. »
Norman Vincent Peale

Remerciements

Ce stage a été effectué à la Société Générale, au sein de l'équipe ITEC/RRF/PNL/I2R, sous la direction d'Olivier Cathala, et sous l'encadrement d'An Tian. Mes premiers remerciements leur sont dédiés pour l'opportunité qu'ils m'ont donnée de travailler sur ce sujet, cela ne peut que refléter la confiance qu'ils ont su m'accorder, et pour la disponibilité dont ils ont fait preuve tout au long de la période du stage.

J'exprime ma profonde reconnaissance à An Tian pour son encadrement et son suivi régulier du stage, depuis mon arrivée jusqu'à la rédaction de ce manuscrit.

Je tiens également à remercier Emeric Viel et Mehdi Abbes qui étaient toujours prêts à répondre à mes questions et à m'aider.

Ils m'ont fait découvrir ainsi plusieurs aspects passionnants de l'informatique en générale, et du domaine du Big Data en particulier, qui ont créé en moi l'envie d'en savoir plus. Grâce à la confiance de toutes ces personnes j'ai pu m'accomplir dans mes missions.

Finalement, je remercie toute l'équipe I2R pour son accueil chaleureux.

Résumé

Les travaux de ce stage ont été effectués au sein de l'équipe ITEC/RRF/PNL/I2R à la Société Générale et portent sur deux volets principaux :

- La réalisation de tests de performance dans le cadre de la mise en place d'une nouvelle architecture s'appuyant sur les technologies Big Data.
- La mise en place d'un service pour le chargement et le requêtage des données issues de la certification et contrôle en utilisant des technologies Big Data.

Mots-clés : Big Data, Hadoop, Spark, Parquet, Kafka, HBase, Hive.

Abstract

This internship was realized in the ITEC/RRF/PNL/I2R team at Société Générale, it deals with two different subjects:

- Carry out performance tests related to a new application that should be put in place to deal with large volumes of data using Big Data technologies.
- Provide a service to allow loading and requesting certification and control data using Big Data technologies.

Keywords: Big Data, Hadoop, Spark, Parquet, Kafka, HBase, Hive.

Table des matières

Liste des tableaux

Liste des figures

Glossaire et sigles utiles

GBIS	Global Banking & Investor Solutions, Banque de Grande Clientèle et Solutions Investisseurs
ITEC	Information Technologies, Technologies de l'Information
RRF	Risk, Referential & Finance, risque, référentiel et finance
PNL	Profits and Losses
FRTB	Fundamental Review of the Trading Book
PTF	Portfolio, Portefeuille
HDFS	Hadoop Distributed File System, Système de Fichiers Distribué de Hadoop
YARN	Yet Another Resource Negotiator
VM	Virtual Machine, machine virtuelle

INTRODUCTION

De nouvelles réglementations FRTB impliquent la production et certification de nouvelles métriques très volumineuse, à savoir soixante milliard de montants par jour, ce qui est équivalent à six térabits par jour. Un changement d'architecture est donc nécessaire pour gérer ces nouveaux volumes en s'appuyant sur les technologies Big Data.

L'équipe ITEC/RRF/PNL/I2R réalise ainsi des tests de performance de composants qui ont été développés dans le cadre de cette nouvelle architecture.

D'autre part, les données issues de la certification et le contrôle de ces métriques deviennent difficiles à visualiser et à analyser à cause de la grande augmentation en volume de données entrainée par les nouvelles contraintes réglementaires (FRTB et autres). L'utilisation des technologies Big Data semble donc être la solution la plus pertinente.

Ce stage a donc pour double objectif de contribuer à la réalisation de tests de performance menés par l'équipe d'une part, et mettre en place un service permettant le chargement et le requêtage des données d'autre part.

J'aborderai dans la première partie du mémoire les différentes technologies Big Data que j'ai eu l'occasion de manipuler, ainsi que quelques concepts clés liés au métier de la finance.

Ensuite, dans la deuxième partie, je présenterai la partie des tests dont j'ai pu contribuer à la réalisation.

Dans la dernière partie, j'entamerai le projet principal sur lequel j'ai travaillé pendant mon stage, à savoir le projet Sentinel qui vise à fournir un outil d'intégration et de requêtage des données en s'appuyant sur des technologies Big Data.

CHAPITRE I : PRESENTATION DE L'ORGANISME D'ACCUEIL

Le groupe Société Générale

Un des tout premiers groupes de services financiers de la zone euro. Il constitue une équipe de 148 300 collaborateurs, présents dans 76 pays dont plus de 60% hors de France métropolitaine.

Le groupe Société Générale est présent dans 3 grands métiers :

- **Banque de détail en France :** 1ère banque en ligne, 2ème banque commerciale pour les grandes entreprises avec 11 M de clients, 3ème banque de détail.
- **Banque de détail et Services Financiers Internationaux :** parmi les principales banques européennes avec 22M de clients.
- **Banque de Grande Clientèle et Solutions Investisseurs :** parmi les leaders européens et mondiaux en marché de capitaux en euro, produits dérivés et financements structurés avec 5000 clients entreprises et institutions financières.

ITEC

ITEC est le pôle informatique de GBIS, qui a pour objectif de fournir des services de production de qualité de manière contrôlée et rentable, et de produire des systèmes d'applications compétitifs, qui permettent à GBIS de développer son activité sur les marchés existants et nouveaux, de lancer de nouveaux produits et de saisir de nouvelles opportunités lorsqu'elles se présentent.

Les missions d'ITEC :

- **Produire** des services et solutions informatiques qui permettent à GBIS de réaliser ses objectifs commerciaux.
- **S'assurer** d'un service de production rentable, de haute qualité et rigoureusement contrôlé à destination des systèmes d'applications commerciales et des infrastructures informatiques.

- **Faciliter** la pénétration de nouveaux marchés, la conformité réglementaire, la croissance des volumes d'activité et l'amélioration de l'efficacité opérationnelle de l'environnement global de traitement, de bout en bout.

ITEC/RRF

ITEC/RRF définit les stratégies IT et garantit la livraison des projets et la maintenance de tous les logiciels liés aux activités de gestion des risques, de référentiels et de financement.

ITEC/RRF/PNL

Il s'agit du département en charge du développement et du support des applications qui s'occupent de la gestion de P&L (Profits and Losses) et RISK/Valuation Datawarehouse.

ITEC/RRF/PNL/I2R

I2R est une des principales applications d'ITEC/RRF/PNL, l'équipe qui s'occupe de cette application s'appelle aussi I2R.

J'ai effectué mon stage au sein d'une sous équipe de I2R qui travaille sur des projets Big Data liés à cette application.

CHAPITRE II : CONCEPTS CLES

Dans ce chapitre, je présenterai les concepts clés, fonctionnels et techniques, qui aideront à la compréhension de la suite du document.

Technologies Big Data

Tout d'abord, je présenterai brièvement Hadoop. Il s'agit du premier outil que l'on cite quand on parle de Big Data.

Hadoop

Hadoop est un Framework libre et open source écrit en Java qui a pour objectif de rendre la création des applications distribuées plus facile, que ce soit par rapport au stockage des données ou par rapport à leur traitement, et scalables. Cela en permettant aux applications de tourner sur des milliers de nœuds et des pétaoctets de données. Chaque nœud est constitué ainsi de machines standards regroupées en grappe. Tous les modules de Hadoop sont conçus en prenant en considération l'idée fondamentale qui dit que les pannes matérielles sont fréquentes et que, par conséquent, elles doivent être gérées automatiquement par le Framework.

Hadoop a été inspiré par la publication de MapReduce, GoogleFS et BigTable de Google. Hadoop a été créé par Doug Cutting et fait partie des projets de la fondation logicielle Apache depuis 2009.

Le noyau d'Hadoop comporte une partie pour le stockage des données : HDFS, et d'une partie pour le traitement des données appelée MapReduce. Hadoop fractionne les fichiers en gros blocs et les distribue à travers les nœuds du cluster. Pour traiter les données, Hadoop transfère le code à chacun des nœuds du cluster, puis chaque nœud traite les données dont il dispose. Cela permet de traiter l'ensemble des données plus rapidement et plus efficacement.

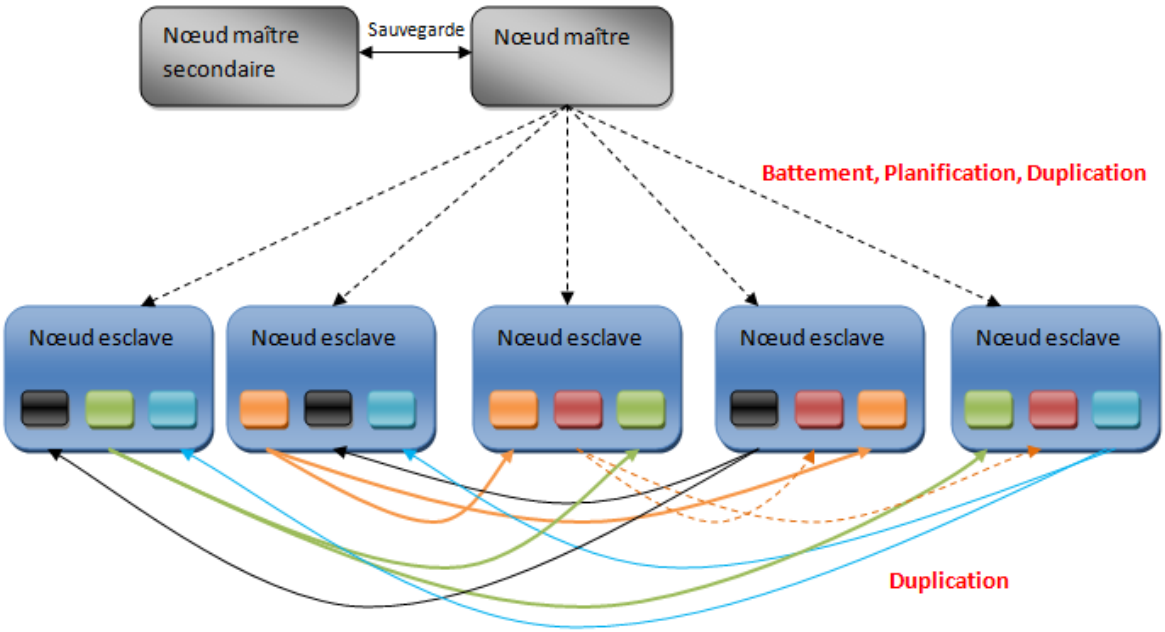
Le Framework Hadoop de base comporte les modules suivants:

- Hadoop Common : les services communs qui servent de support pour les autres modules d'Hadoop.

- Hadoop Distributed File System (HDFS) : un système de fichiers distribué qui permet l'accès avec un débit élevé aux données de l'application.

Le schéma suivant montre le principe général de HDFS :

Figure 1 ✔ Schéma du principe de HDFS



SOURCE : HTTPS://FR.WIKIPEDIA.ORG/WIKI/HADOOP

- Hadoop YARN : un Framework pour la gestion des ressources et la programmation des jobs.

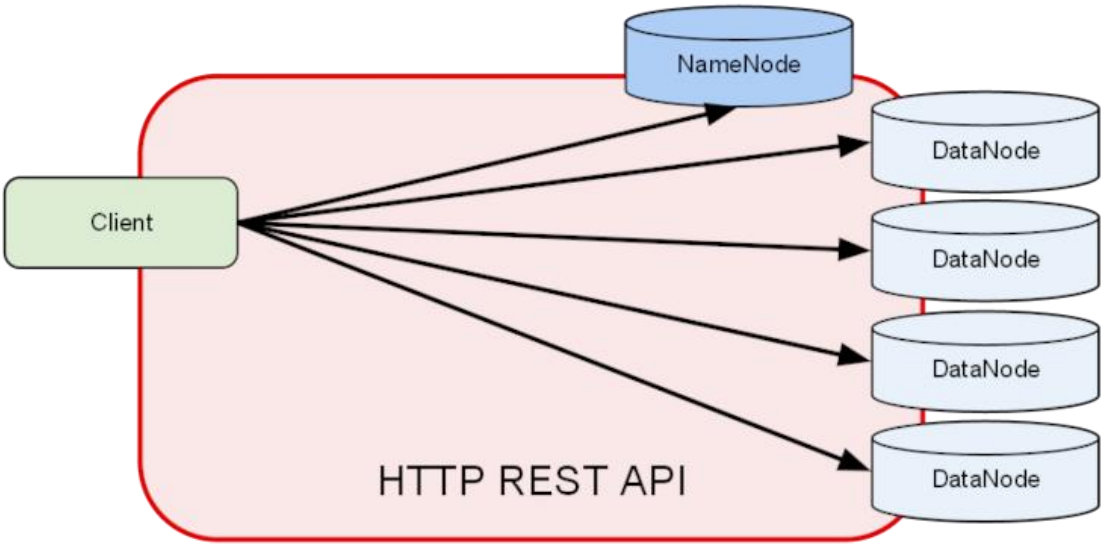
- Hadoop MapReduce : un système, basé sur YARN, permettant de traiter de gros blocs de données en parallèle.

Le terme Hadoop réfère non seulement aux modules de base ci-dessus, mais aussi à son écosystème et à l'ensemble des logiciels qui viennent s'y connecter comme Apache Spark, Apache Hive, Apache HBase et Apache Oozie, pour ont été conçu et développé pour compléter Hadoop et répondre à des besoins plus spécifiques.

WebHDFS

C'est une API HTTP REST qui supporte l'interface du système de fichier complète de HDFS, la figure ci-après en est une représentation simplifiée.

Figure 2 ✔ L'API REST WebHDFS



SOURCE : HTTP://TAGOMORIS.HATENABLOG.COM/ENTRY/20120102/1325466102

Elle offre ainsi une correspondance entre toutes les opérations que l'on peut effectuer via l'API et les méthodes du système de fichier.

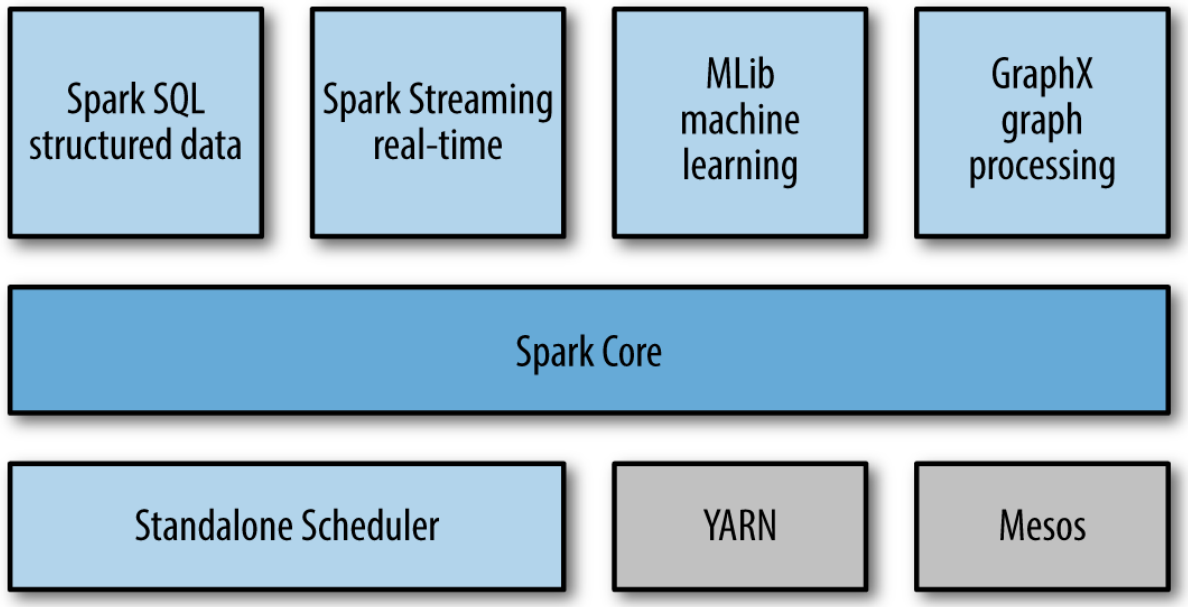
Pour en savoir plus sur les opérations et les méthodes du système de fichier correspondantes, voir[i].

Apache Spark

Spark est Framework de calcul distribué, désigné pour être rapide et général. Il fournit un modèle de programmation simple et complet supportant une gamme vaste d'applications.

La figure ci-dessous montre les différents component Spark :

Figure 3 ✔ Les components de Spark



SOURCE : LEARNING SPARK, HOLDEN KARAU, ANDY KONWINSKI, PATRICK WENDELL AND MATEI ZAHARIA

Pour plus d'informations concernant les différents components de Spark, voir[ii].

Apache Hive

Il s'agit d'une infrastructure d'entrepôt de donnée intégrée sur Hadoop permettant de faciliter la lecture, l'écriture et la gestion des données.

Hive prend en charge l'analyse de grands ensembles de données stockées de façon distribuée en utilisant SQL.

Il fournit un langage similaire à SQL appelé HiveQL avec le schéma lors de la lecture et de manière transparence convertit les requêtes selon différentes forme de moteurs tels que MapRreduce et job Spark. Tous les moteurs d'exécution peuvent fonctionner sur YARN.

Hive prend en charge différents format de fichiers dont le format Parquet via un plugin dans les versions ultérieures et nativement dans les versions récentes.

Pour plus d'informations sur Hive et le langage HiveQL, voir[iii].

Apache Parquet

C'est un format de stockage par colonne compressé et efficace disponible pour tout projet tournant sur l'écosystème Hadoop, indépendamment du choix du Framework de calcul, du modèle de données ou du langage de programmation.

Apache Parquet supporte plusieurs Framework et langages de description de données dont Apache Hive.

Pour plus de détails concernant le format Parquet, voir[iv].

Apache HBase

HBase est un système de gestion de bases de données non-relationnelles (SGBD) distribué, écrit en Java, disposant d'un stockage structuré pour les grandes tables.

C'est une base de données orientée colonne. Basées sur une architecture maître/esclave, les bases de données de ce type sont capables de gérer d'énormes quantités de données, à savoir plusieurs milliards de lignes X plusieurs millions de colonnes. Elle s'installe généralement sur le système de fichiers HDFS d'Hadoop pour faciliter la distribution, même si ce n'est pas obligatoire.

HBase est la base de données la plus adaptée lorsque l'on a besoin d'un accès aléatoire (Random), ou en temps réel à des données présentant une grande volumétrie.

Pour connaître plus sur HBase, voir[v].

Apache Oozie

Oozie est un logiciel de la Fondation Apache servant à l'ordonnancement des « workflows » permettant la gestion des jobs destinés à être exécutés dans l'écosystème Hadoop.

Les « Workflows » dans Oozie sont définis comme une collection de flux de contrôle et d'actions dans un Graphe orienté acyclique. Oozie prend en charge différents types d'actions dont Hadoop MapReduce, les jobs Spark, les requêtes Hive, SSH, et l'envoi email. Oozie peut également être étendu pour supporter d'autres types d'actions.

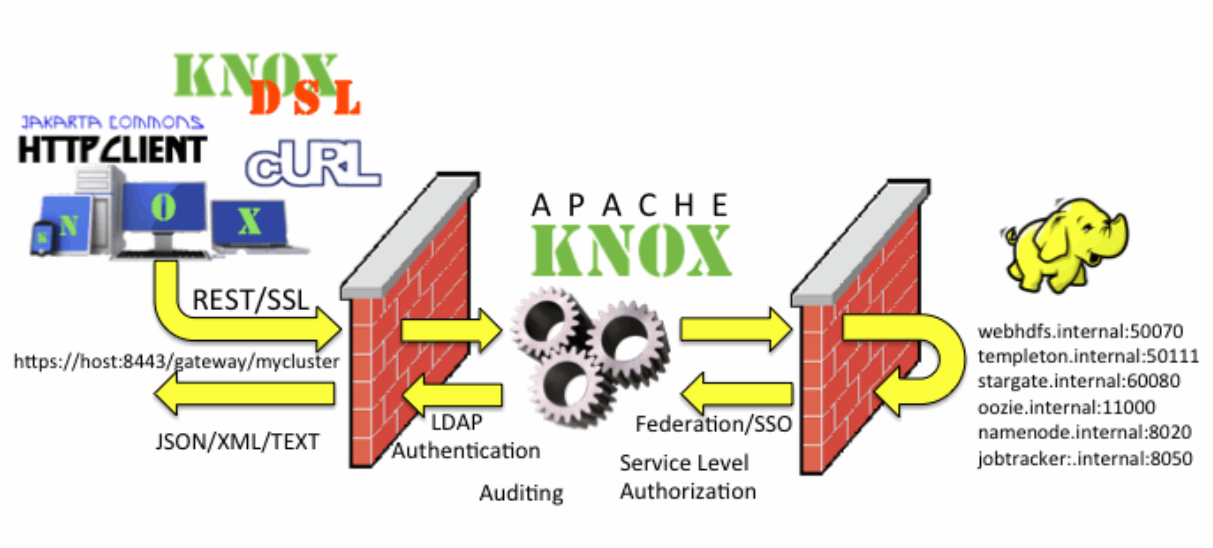
Pour en savoir plus sur Oozie, voir[vi].

Apache Knox

Knox est une API REST qui permet l'accès à l'écosystème Hadoop. Cette passerelle permet un seul point d'accès pour toutes les interactions REST avec les clusters Hadoop. Avec cette capacité, la passerelle Knox est apte à offrir une fonctionnalité très importante en termes de contrôle, d'intégration, de monitoring et d'automatisation des besoins critiques, qu'ils soient administratifs ou analytiques, de l'entreprise. En effet, Knox constitue un complémentaire pour les clusters sécurisé avec Kerberos, voir[vii].

La figure ci-dessous montre l'interaction de Knox avec les différents services de l'écosystème Hadoop.

Figure 4 ✔ La passerelle Knox API REST



SOURCE : HTTPS://KNOX.APACHE.ORG/

Apache Kafka

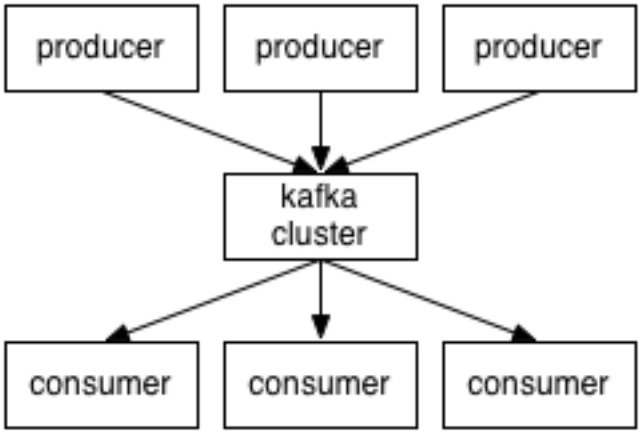
Il s'agit d'un agent de messages développé par la fondation Apache écrit en Scala. Le projet vise à fournir un système unifié, temps réel à latence faible pour la manipulation de flux de données en temps réel. La conception est fortement influencée par les transactions de logs.

Le système Kafka est qualifié de :

- Rapide : un seul Broker Kafka peut gérer des centaines de mégabits de lectures écritures par seconde provenant de plusieurs centaines de clients différents.
- Scalable : car il a été conçu pour permettre à un seul cluster de se comporter comme étant un fédérateur central de données pour une grande organisation. Il peut ainsi être élargi d'une manière élastique et transparente sans temps d'arrêt. Les flux de données sont partitionnés et propagés sur un cluster de machines pour permettre de gérer des flux de données qui dépassent la capacité d'une seule machine d'une part, et pour permettre de constituer des clusters à partir de Consumers coordonnés.
- Durable : les messages sont persistés en disque et sont dupliqués sur le cluster pour éviter la perte de données. Chaque Broker peut gérer des térabits de messages sans impacter la performance.
- Conçu pour être distribué : la conception de Kafka est une conception basée sur un cluster central ce qui permet une forte durabilité et garantit une tolérance aux pannes.

La figure suivante montre le processus par lequel les Producers envoient des messages à travers le réseau au cluster Kafka qui, à son tour, se charge de les servir aux Consumers.

Figure 5 ✔ Fonctionnement de Kafka vu de haut niveau



SOURCE : HTTP://KAFKA.APACHE.ORG/DOCUMENTATION.HTML#INTRODUCTION

Pour plus d'informations sur Apache Kafka et son fonctionnement, voir la documentation officielle[v iii].

Notions liées au métier de la finance

PTF

Désigne une collection d'actifs financiers détenus par un établissement ou un individu. Ce peut aussi désigner des valeurs mobilières détenues à titre d'investissements, de dépôt, de provision ou de garantie.

Deal

Désigne une transaction financière.

Scénario

Désigne une journée de trading de marché pour un PTF donné.

Currency

Désigne la monnaie de change (EURO par défaut).

CHAPITRE III : TESTS DE PERFORMANCE

À mon arrivée à la Société Générale, l'équipe I2R Big Data, avec qui j'étais amenée à travailler, faisaient des tests de performance de plusieurs composants qu'ils ont mis en place, en vue de s'assurer que les temps de réponse correspondent à ce que l'on attendait, ainsi que étudier les possibilités d'améliorations de ces performances en faisant varier des paramètres liés aux technologies Big Data utilisées.

Dans ce chapitre, je citerai le premier sujet sur lequel j'ai travaillé pendant mon stage, à savoir une partie des tests de performance que mon maître de stage était amené à faire, et dont j'ai contribué à la réalisation.

FRTB Parquet Search API

Parmi les composants qui ont été développés, et dont le client est une équipe de ITEC/RRF/PNL, est une API « FRTB Parquet Search API ». Cette API permet de zoomer sur un PTF et chercher, dans ce PTF, un ou plusieurs scénarios en précisant la « currency ». Les données en entrée sont, comme le nom de l'API l'indique, stockées en Parquet et partitionnées par PTF. Le résultat de la recherche est exporté en CSV sous 2 format différents, à passer en paramètre : le format ligne et le format colonne. Le but est de comparer la performance des deux formats.

L'objectif des tests, dans leur globalité, est d'étudier la performance de cette API en la comparant à une autre dont les données en entrée sont stockées dans HBase. Ensuite, tester la performance de l'API face à des appels concurrents.

La partie sur laquelle j'ai travaillée consistait à effectuer des appels parallèles à l'API (ce qui a déjà été fait pour HBase), ce qui revient à lancer des jobs simultanés sur le cluster et étudier les variations du temps d'exécution en fonction du nombre d'appels.

Données en entrée

Les données que l'on utilise sont des données de risques de marché stockées sur le HDFS, en Parquet, partitionnées par PTF.

Environnement des tests

Chaque accès à l'API revient à lancer un job Spark qui passe par deux étapes : une pour exécuter la requête afin de chercher les informations souhaitées, et la deuxième pour exporter le résultat en CSV.

On fait tourner les tests sur un cluster Hadoop sécurisé avec Knox, et on note le temps d'exécution.

Paramètres

Les paramètres fonctionnels :

Portefeuille : le portefeuille que l'on sélectionne pour effectuer la recherche.

Scénario : le (les) scénario (s) que l'on souhaite extraire.

Currency : la valeur par défaut est EURO.

Les paramètres techniques :

Queue : pour mieux gérer la répartition des ressources entre les différentes équipes qui utilisent le cluster, une queue a été mise en place pour chaque équipe.

Format : le résultat de l'export est en CSV suivant deux formats différents :

- Colonne :

Table 1 ✔ Format d'export CSV en colonne

CSV format Ligne											
deal1	currency1	scenario1	v1	v2	v3	errorflag	scenario2	v1	v2	v3	errorflag
deal1	currency2	scenario1	v1	v2	v3	errorflag	scenario2	v1	v2	v3	errorflag
deal2	currency1	scenario1	v1	v2	v3	errorflag	scenario2	v1	v2	v3	errorflag

- Ligne :

Table 2 ✔ Format d'export CSV en ligne

CSV format colonne						
deal1	scenario1	currency1	v1	v2	v3	errorflag
deal1	scenario1	currency2	v1	v2	v3	errorflag
deal1	scenario2	currency1	v1	v2	v3	errorflag
deal1	scenario2	currency2	v1	v2	v3	errorflag
deal2	scenario1	currency1	v1	v2	v3	errorflag
deal2	scenario2	currency1	v1	v2	v3	errorflag

Les valeurs de v1, v2 et v3 sont définies comme suit :

[v1, v2, v3, errorflag] = [deformedValue, inCurrency, inEuro , errorflag]

- Comparaison des deux formats :

Le stockage en colonne est moins volumineux que le stockage en ligne, car les champs sont moins redondants (on ne répète pas le deal tant de fois qu'on a de scénarios).

En revanche, pour générer le CSV, il faut pour chaque ligne chercher tous les scénarios liés à un deal, par conséquent, le stockage en colonne est plus lent que le celui en ligne.

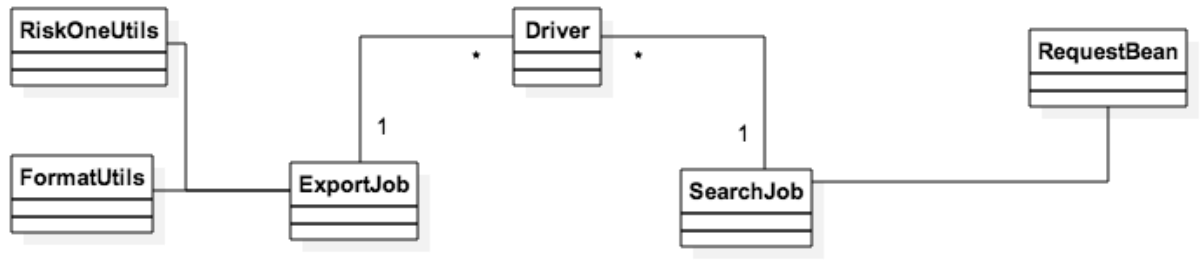
Autres paramètres : En plus de paramètres supplémentaires liés à la mémoire dédiée à l'exécution du job.

Conception de l'API

L'API a été conçue est développée par mon maître de stage avant mon arrivée dans l'équipe, mais il m'était crucial de bien assimiler le code de l'API et son architecture afin de comprendre son fonctionnement.

La figure ci-dessous montre le diagramme de classes de l'API :

Figure 6 ✔ Diagramme de classes de l'API FRTB Parquet Search



SearchJob : la partie du code qui sert à effectuer la recherche souhaitée.

ExportJob : la partie du code qui se charge de sauvegarder le résultat en CSV suivant le format précisé.

Driver : la classe principale qui effectue la recherche et sauvegarde le résultat.

RiskOneUtils et **FormatUtils** : ce sont des classes utilitaires.

RequestBean : elle sert à définir la requête correspondant à la recherche à effectuer.

Résultats

J'ai lancé des appels concurrents à l'API progressivement, et j'ai noté le temps d'exécution du job (jusqu'à l'écriture du fichier CSV résultant).

Deux séries de tests ont été effectuées, une en faisant des appels pour requêter le même PTF, auquel cas j'en ai utilisé le plus grand, et l'autre en appelant des PTF différents, auquel cas je faisais un appel au plus grand et tous les autres appels à un PTF moyen.

Pour tous les tests, les scénarios à rechercher ont été au nombre de quatre, en utilisant la Currency par défaut.

Les résultats sont présentés dans le tableau suivant :

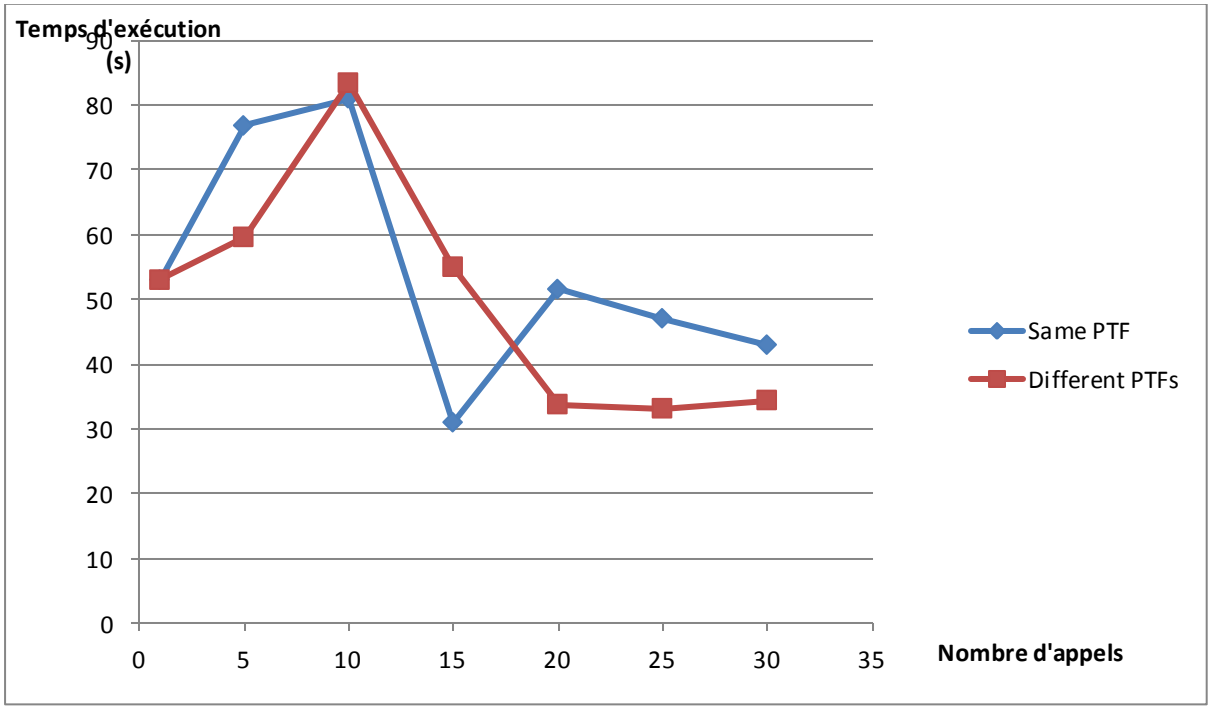
Table 3 ✔ Tableau des résultats des tests d'accès concurrents

Nombre d'appels	Même PTF (seconds)	PTFs fdifférents (seconds)
1	53	53
5	76.8	59.6
10	81	83.4
15	31	55
20	51.55	33.8
25	47.04	33.16
30	42.97	34.43

La valeur notée pour le temps d'exécution correspond à la valeur moyenne du temps d'exécution de tous les appels.

Le graphe suivant montre l'évolution du temps d'exécution en fonction du nombre d'appels :

Figure 7 ✔ Évolution du temps d'exécution de la recherche et extraction en fonction du nombre d'appels



Analyses

Le graphe est loin de ce que l'on attendait. En fait, on s'attendait à ce que le temps d'exécution augmente au fur et à mesure que l'on incrémente le nombre d'appels.

Après une longue discussion avec tous les membres de l'équipe, il s'est avéré que nous ne possédons pas de gestionnaire d'appels concurrents au cluster Hadoop. Autrement dit, nous ne pouvons pas se baser sur les résultats de ces tests, car, le temps d'exécution dépend de la disponibilité des ressources sur le cluster.

Par conséquent, tous ces tests ont juste permis de vérifier que l'on a besoin d'un serveur qui permet de gérer les accès concurrents au cluster pour pouvoir reproduire un graphe qui soit exploitable, et sur lequel on peut se baser pour arriver à des conclusions logiques.

Étant donné que ce sujet n'était pas d'urgence, et que je devais avancer sur le projet principal de mon stage, que je présenterai dans le prochain chapitre, il a été décidé de s'arrêter à ce stade pour ces tests.

Par la suite j'étais amenée aussi à refaire une partie des tests qui ont été effectués par mon encadrant sur le plus grand PTF (EURUS) d'une part, dans le but de sauvegarder les fichiers CSV en sortie, et d'autre part à faire de même pour un PTF moyen (FXSCA), pour pouvoir comparer le deux.

Les résultats sont présentés dans le tableau suivant :

Table 4 ✔ Tableau de comparaison des résultats de tests de performance entre deux PTFs.

PTF	Taille Parquet d'origine	NB de scénarios requêtés	CSV (scénario en colonnes)		CSV (scenario en lignes)	
EURUS	442.6 M	1	5M	**1min27s**	5M	**0min54s**
		2	8M	**1min39s**	10M	**1min1s**
		4	15M	**1min34s**	20M	**1min7s**
		8	29M	**1min37s**	40M	**0min56s**
		16	58M	**1min45s**	80M	**1min1s**
		45	155M	**1min56s**	220M	**1min4s**
		75	274M	**2min11s**	384M	**1min9s**
		100	351M	**2min20s**	492M	**1min11s**
		1500	4G	**16min30s**	6G	**3min43s**
FXSCA	160.7 M	1	666.3 K	**0m56s**	874.6 K	**0m44s**
		2	1.0 M	**0m56s**	1.7 M	**0m44s**
		4	1.8 M	**0m56s**	3.4 M	**0m44s**
		8	3.3 M	**0m56s**	6.7 M	**0m44s**
		16	3.3 M	**0m56s**	6.7 M	**0m44s**
		45	3.3 M	**0m56s**	6.7 M	**0m44s**
		75	3.3 M	**0m56s**	6.7 M	**0m44s**
		100	3.3 M	**0m56s**	8.4 M	**0m44s**
		1500	1.3 G	**0m56s**	2.5 G	**0m44s**

On a eu une réunion avec les clients pour discuter les résultats qui semblaient être logiques, et on leur a fourni le chemin HDFS dans lequel se trouvent les fichiers CSV générés.

Tests Kafka

Pendant la première partie de mon stage, j'ai pu aider mon maître de stage à réaliser des tests de performance de publication de données sur des serveurs Kafka.

Contexte des tests

Le « Expected Shotfall » (déficit prévue) causé par les nouvelles réglementations FRTB va engendrer une grande augmentation du volume de données. L'envoie des fichiers CSV de Highway[1] à I2R avec l'architecture existante constitue une problématique et l'utilisation de Kafka semble être une meilleure alternative.

De point de vue technique, les tests dont j'ai contribué à la réalisation ont été effectués sur des serveurs VM Kafka, et on a utilisé des VMs pour simuler les Producers de Highway.

On dispose de moins de VMs que de Producers Highway, mais on utilise plusieurs Producers par VM pour remédier à ce problème.

Les tests ont été effectués en utilisant des clients JAVA, .Net et IKVM.

On envoie des messages (données) depuis les Producers, et on note la durée entre l'envoi et la réception de ces données par les Brokers Kafka.

Paramètres

Pour cette série de tests, les paramètres dont dépend la performance sont :

- Nombre de Producers : on incrémente le nombre de Producers qui envoient les données progressivement, on peut utiliser au maximum quatre Producers par VM (si on dépasse quatre, cela fait planter la VM).

- Ack : désigne le nombre d'accusés de réception que le Producer exige que le leader ait reçus pour qu'il considère que la requête est complète.

  Tous les tests ont été fait en utilisant une valeur de ack égale à 1 puis égale à ALL.

  Le tableau suivant explique la différence entre ack = 1 et ack = ALL :

[1] Highway est une application qui envoie des données à I2R

Table 5 ✔ Différence entre ack = 1 et ack =ALL

Ack = 1	Cela signifie que le leader va écrire l'enregistrement dans son log en local, mais il va répondre sans attendre qu'il ait reçu tous les accusées de réception de la part des suiveurs. Dans ce cas, si le leader tombe en panne juste après avoir accusé la réception du message mais avant que les suiveurs l'aient dupliqué, le message sera perdu.
Ack = ALL	Cela signifie que le leader va attendre que tous les suiveurs aient accusés la réception de l'enregistrement. Ce qui garantie que le message ne serait pas perdu tant que le dernier suiveur fonctionne. Il s'agit de plus forte garantie.

Pour en savoir plus sur la différence entre les différentes valeurs que peut prendre ce paramètre, voir la documentation officielle de Kafka.

Tous les messages ont une taille fixe (13 million messages), donc le nombre de messages est à multiplier par le nombre de Producers ou d'instances de VMs utilisées.

Pour comparer la performance de chacun des clients Java, IKVM[2] et .Net, on calcule la vitesse d'envoi des messages :

Vitesse = Nombre de messages / temps d'envoi

L'utilisation de IKVM est due au fait que l'on ne dispose pas de Driver .Net compatible avec un serveur Kafka sécurisé. De ce fait, on a utilisé IKVM pour convertir du code .Net en byte code Java, que l'on veut comparer avec le vrai client Java, en attendant d'avoir un Driver .Net qui fonctionne.

Le serveur Kafka VM est non sécurisé, par conséquent on a pu comparer le client issu d'IKVM avec le client .Net aussi.

Les résultats obtenus pour le Producer Java VM sont présentés dans le tableau suivant :

[2] IKVM est une implémentation de Java pour Microsoft .NET Framework. Il permet de convertir du code .NET en byte code Java.

Table 6 ✔ Tableau des résultats des tests Kafka pour le Producer Java VM

Kafka Producer						
	Producer	ack	Message	Instance	Time	Speed
Topic Partition : 3 Replication : 3	Java VM	**1**	13 Million	1	21s	619 000/s
			26 Million	2	25s	1 040 000/s
			39 Million	3	27s	1 444 000/s
			52 Million	2+2=4	32s	1 625 000/s
			65 Million	2+3=5	36s	1 806 000/s
			78 Million	3+3=6	37s	2 108 000/s
			91 Million	3+3+1=7	38s	2 395 000/s
			104 Million	3+3+2=8	42s	2 476 000/s
			117 Million	3+3+3=9	39s	3 000 000/s
			130 Million	3+3+3+1=10	40s	3 250 000/s
			143 Million	3+3+3+2=11	40s	3 575 000/s
			156 Million	3+3+3+3=12	40s	3 900 000/s
			792 Million	3+3+3+3=12	264s	3 000 000/s
	Java VM	**ALL**	13 Million	1	55s	236 000/s
			26 Million	2	83s	313 000/s
			39 Million	3	87s	448 000/s
			52 Million	2+2=4	103s	505 000/s
			65 Million	2+3=5	113s	575 000/s
			78 Million	3+3=6	118s	661 000/s
			91 Million	3+3+1=7	125s	728 000/s
			104 Million	3+3+2=8	146s	712 000/s
			117 Million	3+3+3=9	151s	775 000/s
			130 Million	3+3+3+1=10	169s	769 000/s
			143 Million	3+3+3+2=11	175s	817 000/s
			156 Million	3+3+3+3=12	176s	886 000/s
			480 Million	3+3+3+3=12	548s	876 000/s

Le tableau ci-après montre les résultats pour le Producer IKVM VM:

Table 7 ✔ Tableau des résultats des tests Kafka pour le Producer IKVM VM

Kafka Producer						
	Producer	ack	Message	Instance	Time	Speed
Topic Partition : 3 Replication : 3	IKVM VM	**1**	13 Million	1	43s	302 000/s
			26 Million	2	48s	542 000/s
			39 Million	3	49s	796 000/s
			52 Million	2+2=4	48s	1 083 000/s
			65 Million	2+3=5	55s	1 182 000/s
			78 Million	3+3=6	55s	1 418 000/s
			91 Million	3+3+1=7	56s	1 625 000/s
			104 Million	3+3+2=8	58s	1 763 000/s
			117 Million	3+3+3=9	61s	1 918 000/s
			130 Million	3+3+3+1=10	57s	2 281 000/s
			143 Million	3+3+3+2=11	59s	2 424 000/s
			156 Million	3+3+3+3=12	59s	2 644 000/s
			792 Million	3+3+3+3=12	349s	2 269 000/s
	IKVM VM	**ALL**	13 Million	1	61s	213 000/s
			26 Million	2	87s	299 000/s
			39 Million	3	94s	415 000/s
			52 Million	2+2=4	95s	547 000/s
			65 Million	2+3=5	107s	607 000/s
			78 Million	3+3=6	116s	672 000/s
			91 Million	3+3+1=7	129s	705 000/s
			104 Million	3+3+2=8	135s	770 000/s
			117 Million	3+3+3=9	145s	807 000/s
			130 Million	3+3+3+1=10	158s	823 000/s
			143 Million	3+3+3+2=11	168s	851 000/s
			156 Million	3+3+3+3=12	181s	862 000/s
			480 Million	3+3+3+3=12	553s	898 000/s

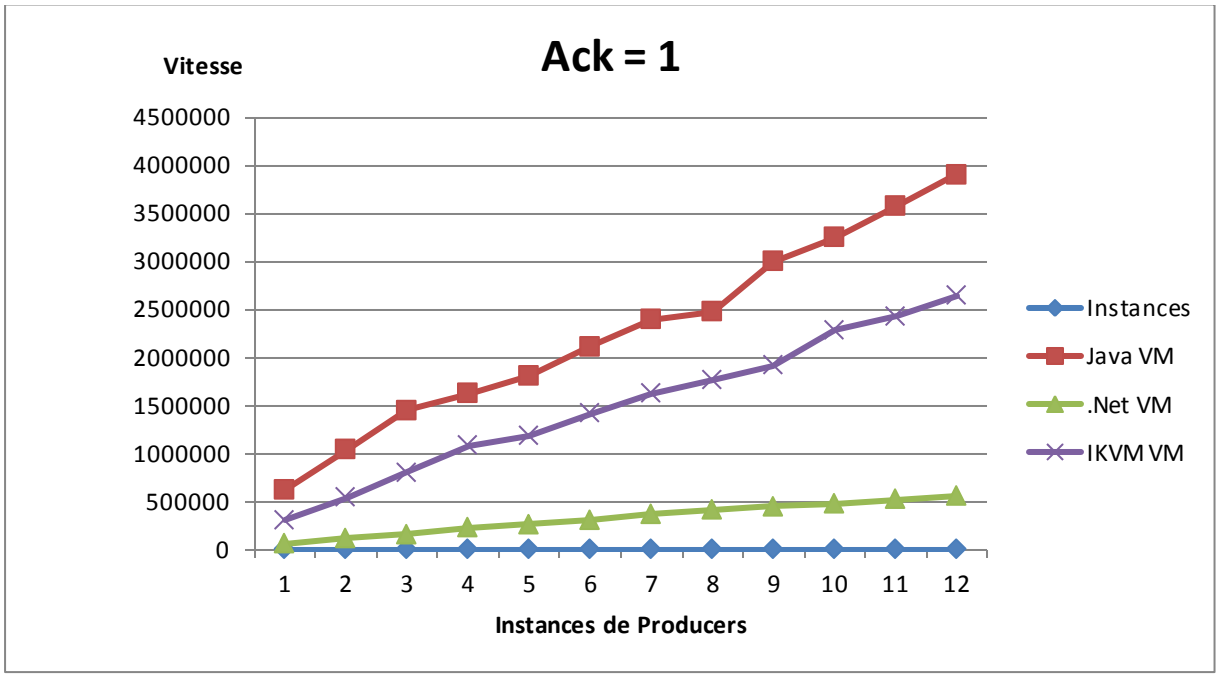
Finalement, pour le Producer .Net VM, les résultats sont montrés dans le tableau suivant :

Table 8 ✔ Tableau des résultats des tests Kafka pour le Producer .Net VM

Kafka Producer						
Topic Partition : 3 Replication : 3	Producer	ack	Message	Instance	Time	Speed
	.Net VM	**1**	13 Million	1	219s	59 000/s
			26 Million	2	231s	112 000/s
			39 Million	3	256s	152 000/s
			52 Million	2+2=4	230s	226 000/s
			65 Million	2+3=5	254s	256 000/s
			78 Million	3+3=6	258s	302 000/s
			91 Million	3+3+1=7	250s	364 000/s
			104 Million	3+3+2=8	253s	411 000/s
			117 Million	3+3+3=9	265s	442 000/s
			130 Million	3+3+3+1=10	274s	474 000/s
			143 Million	3+3+3+2=11	274s	522 000/s
			156 Million	3+3+3+3=12	281s	555 000/s
			480 Million	3+3+3+3=12	796s	603 000/s
	.Net VM	**ALL**	13 Million	1	209s	62 000/s
			26 Million	2	218s	119 000/s
			39 Million	3	232s	168 000/s
			52 Million	2+2=4	238s	218 000/s
			65 Million	2+3=5	243s	267 000/s
			78 Million	3+3=6	254s	307 000/s
			91 Million	3+3+1=7	254s	358 000/s
			104 Million	3+3+2=8	264s	394 000/s
			117 Million	3+3+3=9	264s	443 000/s
			130 Million	3+3+3+1=10	274s	475 000/s
			143 Million	3+3+3+2=11	279s	513 000/s
			156 Million	3+3+3+3=12	286s	545 000/s
			480 Million	3+3+3+3=12	825s	582 000/s

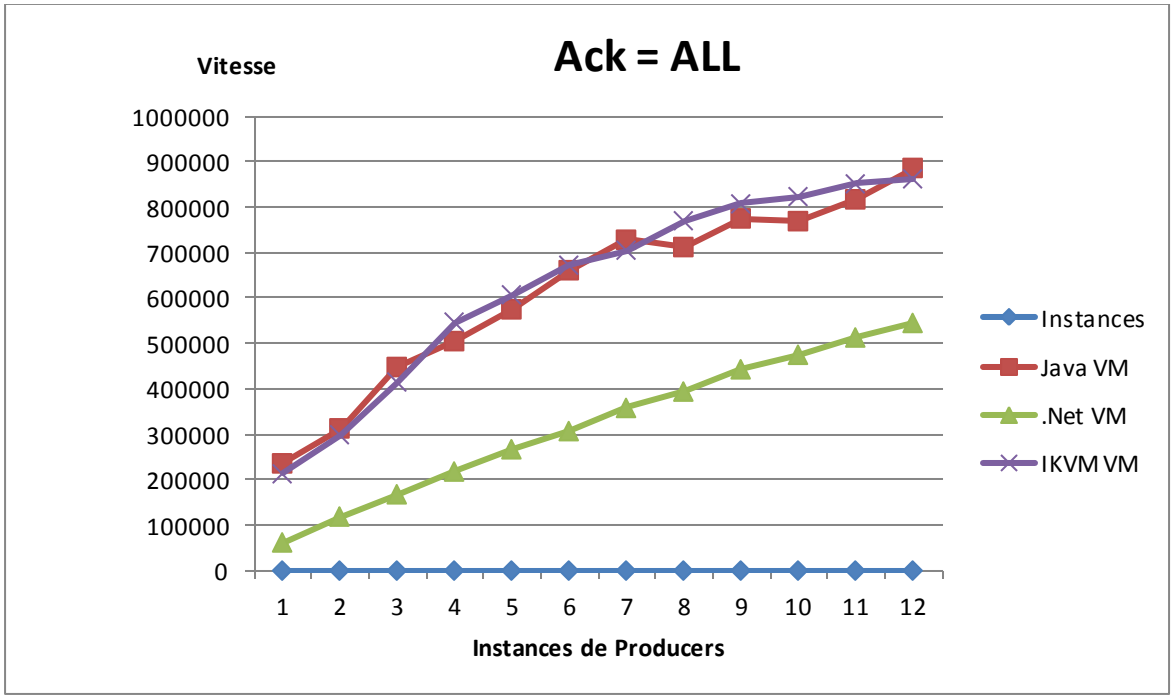
Pour comparer les résultats, on a envisagé de faire des graphes. La figure suivante montre les résultats de la comparaison des Producers Java, IKVM et .Net pour ack égal à 1.

Figure 8 ✔ Comparaison des Producers Java, IKVM et .Net VM pour ack = 1



Pour ack égal à ALL, les résultats sont présentés dans le graphe suivant :

Figure 9 ✔ Comparaison des Producers Java, IKVM et .Net VM pour ack = ALL



Analyses

La conclusion que l'on a pu tirer des résultats est que l'on n'a pas encore atteint la limite en vitesse, et donc on peut augmenter le nombre de messages envoyés en incrémentant davantage le nombre de VMs, et par conséquent le nombre d'instances de Producers.

Ces résultats restent partiels. En effet, des tests avec des vrais Producers Highway ont été prévus pour vérifier leur exactitude. Ils nous ont donc permis de définir les meilleures configurations à utiliser pour les vrais tests.

Conclusion du chapitre

Dans ce chapitre j'ai présenté le premier sujet sur lequel j'ai travaillé pendant les premières semaines de mon stage, à savoir la contribution à la réalisation d'une partie des tests de performance que mon maître de stage était amené à effectuer.

Cela m'a permis de me familiariser avec les technologies Big Data Spark et Kafka et le format Parquet et comprendre la façon dont on peut s'en servir, et aussi avec l'environnement de travail en général.

Pendant cette période, j'ai pu m'intégrer dans l'équipe et m'habituer au rythme de travail, que j'ai trouvé assez accéléré au début. Il m'a fallut ainsi assimiler les notions liées au métier de la finance d'une part, et comprendre le fonctionnement des composantes à tester de point de vue technique pour pouvoir les utiliser d'autre part.

De plus, pendant toute la période du stage, je participais à toutes les réunions que l'équipe était amenée à organiser afin de discuter les résultats des tests ou de les présenter à d'autres équipes, ce qui m'a permis d'enrichir largement mes connaissances techniques sur différentes technologies.

CHAPITRE IV : EXPLORATION DE LA PISTE D'AUDIT SENTINEL EN BIG DATA

J'entamerai dans ce chapitre le sujet principal sur lequel j'ai travaillé pendant mon stage. Je présenterai ainsi le contexte du projet en expliquant le besoin derrière son lancement, la solution que l'on adopté pour répondre à la problématique du projet en Big Data, et toutes les étapes de réalisation que l'on n'a pas encore terminée au jour de la rédaction de ce document.

Contexte du projet

Les données qui proviennent de la piste d'Audit Sentinel, ce sont des contrôles effectués sur les deals de risques de crédits, sont difficiles à visualiser et analyser par MACC[3], et ce à cause de leur volumétrie et leur verbosité.

Ces données, dont on ne peut pas garder l'historique, sont très peu exploitées, parce que pour faire tourner une simple requête, cela prend un temps considérablement long.

Le besoin est donc de pouvoir explorer ces données et en faire des analyses.

Pour répondre à ce besoin, on a opté pour une solution qui consiste à sauvegarder les données en utilisant les technologies liées à l'écosystème Hadoop, et mettre à disposition de MACC un outil leur permettant de les requêter.

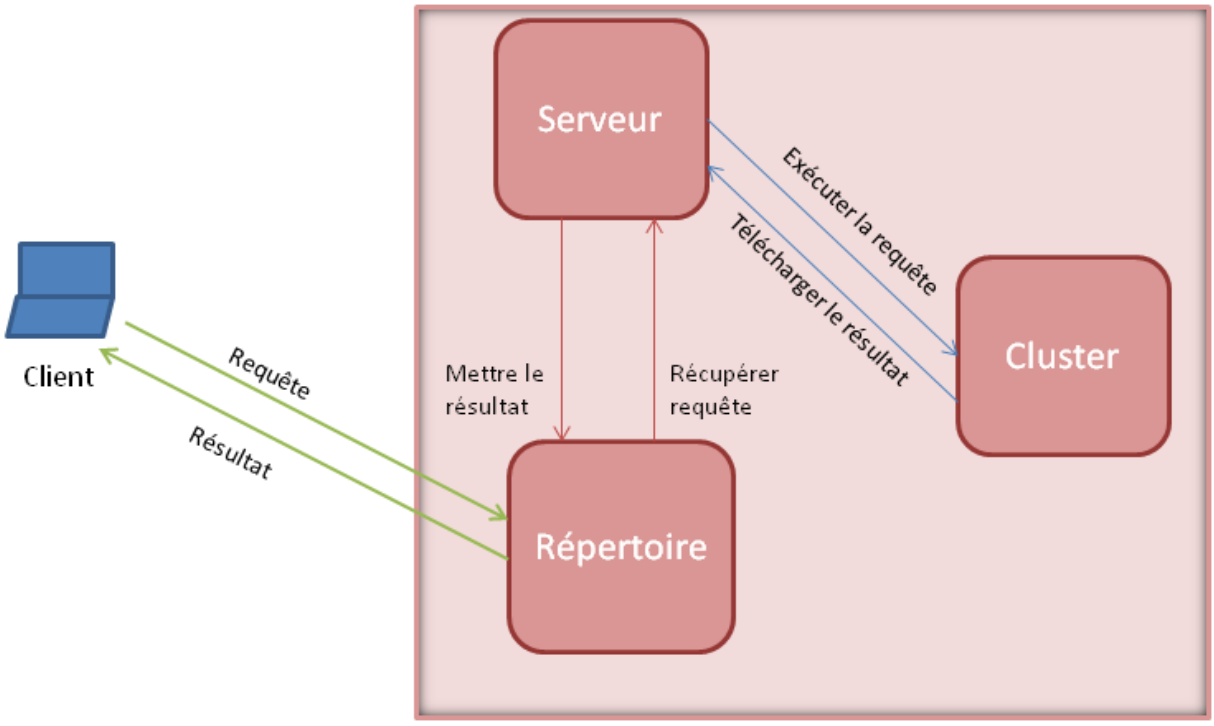
[3] MACC est une unité de la SGCIB

Architecture de la solution

Le service que l'on a utilisé est développé par une autre équipe d'ITEC/RRF/PNL, qui travaille aussi sur des projets Big Data. L'objectif était donc de reprendre ce qui a été fait et l'adapter pour répondre à notre besoin.

La figure ci-dessous représente l'architecture de la solution :

Figure 10 ✔ Architecture de la solution mise en place pour analyser les données Sentinel



Le mode de fonctionnement du service peut être expliqué par les étapes suivantes :

- Le client écrit sa requête SQL dans un fichier texte et sauvegarde le fichier dans le répertoire.
- Le serveur fait tourner un service qui détecte le fichier qui a été créé, récupère la requête et l'exécute sur le cluster, je reviendrai sur cette étape pour l'expliquer plus en détail par la suite.
- Une fois la requête exécutée et le fichier CSV contenant le résultat généré, le serveur le télécharge et le met dans le répertoire pour que le client puisse le récupérer.

Déploiement du service

Comme je l'ai déjà mentionné, il a été décidé d'utiliser un service développé par une autre équipe et l'adapter afin de le faire fonctionner pour notre projet.

J'étais donc amené à récupérer le code source du service, et j'ai eu une réunion avec la personne qui se charge de son développement et son support pour m'expliquer la façon générale dont fonctionne chacun de ses composants.

Il m'a fallut ainsi bien comprendre le code des composants qu'on allait utiliser, et leur mode de fonctionnement.

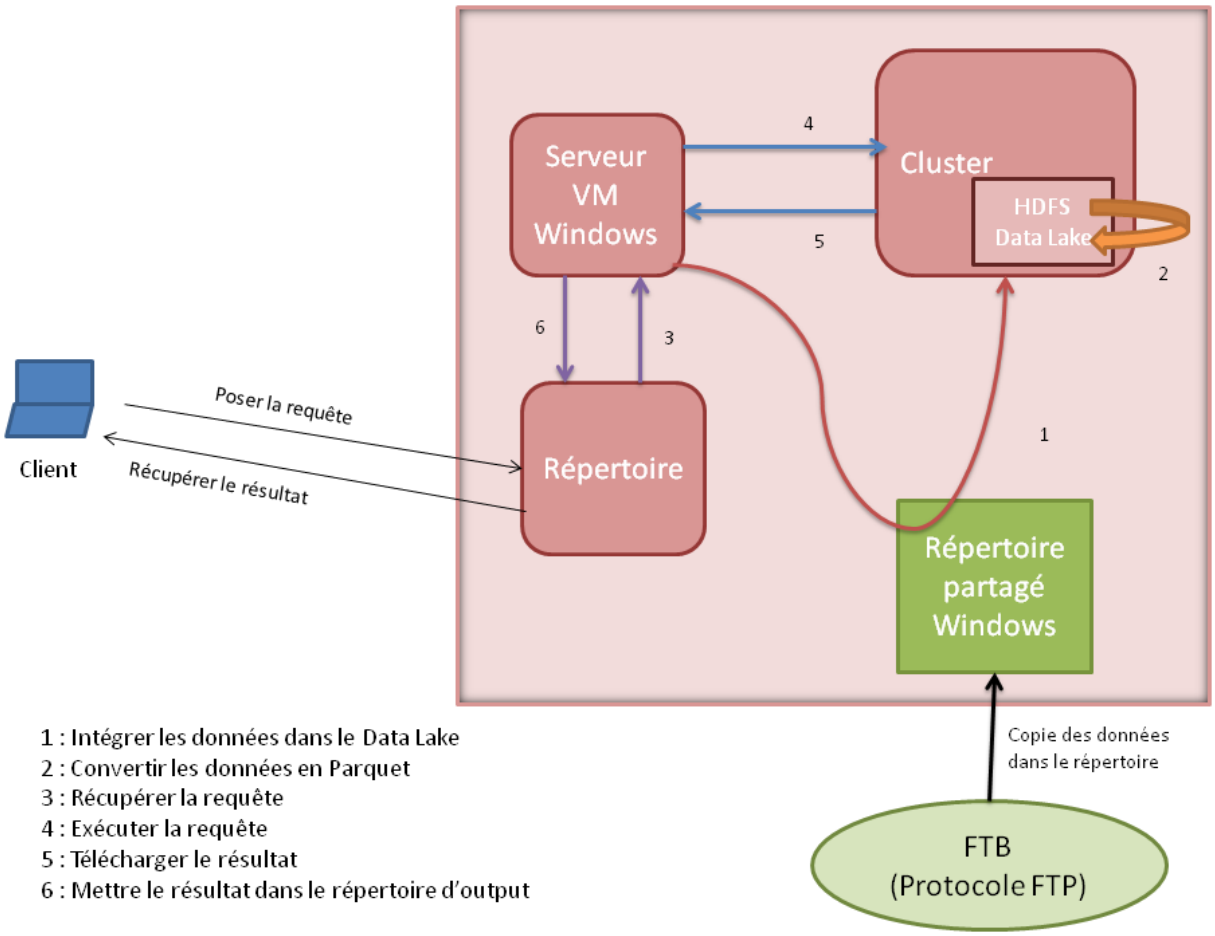
Dans un premier temps j'ai compilé le projet correspondant au service et je l'ai déployé sur ma machine en local, dans le but de s'assurer que les modules dont on a besoin fonctionnent comme ce que l'on attendait.

Ensuite, après une discussion avec mon maître de stage, où il m'a expliqué la façon dont on veut déployer le service, j'ai pu réaliser un diagramme qui montre les différents composants qui ont été mis en place.

La figure suivante montre le diagramme en question, qui met en évidence le processus par lequel passent les données Sentinel, depuis leur chargement dans le Data Lake[4], et les transformations qu'elles subissent, jusqu'à ce qu'elles soient prêtes à requêter par les clients.

[4] Data Lake : il s'agit du nom de l'ensemble des disques permettant un stockage distribué sous HDFS.

Figure 11 ✔ Déploiement du service permettant d'intégrer et requêter les données Sentinel



Dans ce qui suit, j'expliquerai le fonctionnement de chacun de ces composants. Pour ce faire, je vais reprendre le fonctionnement du service étape par étape :

Copie des fichiers CSV

Les données Sentinel sont initialement au format CSV. Nous avons prévu au début de traiter onze fichiers par jour.

La première étape consiste à copier les fichiers CSV dans un répertoire partagé Windows via un protocole FTP.

Nous avons préparé le répertoire qui est supposé recevoir les fichiers et nous avons effectué un test pour s'assurer que l'on reçoit bien les fichiers copiés.

Nous avons aussi fait en sorte qu'au bout de trois jours les anciens fichiers seront effacés pour libérer l'espace aux nouveaux fichiers.

Jusqu'au jour où je rédige ce document, la copie automatique des fichiers quotidiens n'a pas encore été lancée. En revanche, nous disposons de données correspondant à six dates que l'on a reçues en Juillet, ce qui a permis d'avancer sur le projet.

Intégration des données dans le Data Lake

L'intégration des données dans le Data Lake se fait via un service qui tourne sur le serveur VM Windows. La copie des fichiers depuis le répertoire partagé vers le Data Lake se fait via WebHDFS.

Étant donné que l'on reçoit les données une fois par jour à une heure précise, on a opté pour la programmation de l'insertion de ces fichiers dans le Lake : à une heure précise le service se lance et vérifie s'il y a de nouveaux fichiers qui ont été ajoutés et les met dans le répertoire correspondant dans le Lake.

Organisation des fichiers

Les fichiers quotidiens que l'on reçoit correspondent à des tables Sentinel différentes, chaque fichier garde la même nomenclature :une partie liée au nom de la table suivie de la date.

J'ai donc décidé de créer un répertoire par table, par type de fichier, et au moment ou le programme copie le fichier, il crée un répertoire dont le nom correspond à la date du fichier avant de le copier dans ce répertoire. Ci-après un exemple :

- Table1
    - 2016-07-07
    - 2016-07-08
    - 2016-07-11
    - 2016-07-12
    - …
- Table2
- Table3
- Table4
- …

Détection des fichier contenant les requêtes

La détection des fichiers qui contiennent les requêtes à exécuter est assurée par un module qui consiste en un Watcher : un programme que l'on va relancer toutes les dix secondes qui va vérifier l'existence de nouveaux fichiers dans le répertoire, et dès qu'il en trouve un il déclenche la récupération du fichier puis l'extraction de la requête qu'il contient, et enfin l'exécution de cette requête (cette dernière étape sera détaillé dans la partie suivante).

Le service, qui tourne sur le serveur VM Windows, est capable, dans le cas où il trouve plusieurs fichiers, de les mettre dans une queue d'attente, et de les traiter un par un.

Une la requête exécutée, le fichier CSV contenant le résultat est téléchargé via WebHDFS et mis dans le répertoire d'output.

En cas d'erreur, quelle qu'en soit la nature, un fichier contenant l'erreur renvoyée par le serveur est généré à la place du fichier CSV.

Exécution des requêtes

Après avoir récupéré la requête depuis un fichier que le client a mis dans le répertoire, le programme lance un job Spark qui va l'exécuter sur le cluster.

Pour lancer le job Spark sur le cluster, on procédait au début par une connexion SSH. En revanche, cette méthode présente des failles de sécurité et on ne pouvait pas passer en production en l'utilisant, c'est la raison pour laquelle le module a été amélioré. Dans la nouvelle version du service, on utilise Oozie qui se connecte au cluster via Knox. Cette nouvelle méthode est plus sécurisée, et par conséquent on peut mettre le service en production.

Les étapes d'exécution sont les suivantes :

- Le programme lance un job Oozie, qui lance un script Shell permettant d'exécuter le job Spark.

Remarque :

Oozie connaît un bug qui fait que l'on ne peut lancer un job Spark directement. Pour contourner ce problème on a opté pour cette solution qui consiste à lancer un simple script Shell qui contient la commande pour lancer le job Spark.

- La commande Shell prend en paramètre la requête à exécuter et le répertoire d'output. Elle fait appel à un script Python.

- Le script Python exécute la requête sur les données stockées sous forme de tables Hive.

J'avais mentionné auparavant que les données dont on dispose sont au format CSV, et que les requêtes font appel à des tables Hive. Il existe donc une étape intermédiaire qui consiste à convertir les données en Parquet, et créer des tables Hive à partie des données stockées en Parquet. Le choix de conversion des données en Parquet et ne pas utiliser du CSV directement est justifié par le fait que le Parquet est un format compressé : plus que dix fois moins volumineux que le CSV.

Cette étape est expliquée dans la partie prochaine.

Conversion des données en Parquet

Pour développer le composant qui permet de convertir les fichiers CSV en parquet, je suis partie du code qu'avait commencé mon maître de stage lorsqu'on avait qu'une seule table, et je l'ai compléter pour prendre en considération toutes le tables dont on dispose jusqu'au jour de la rédaction de ce manuscrit.

Comme on connaît l'heure à laquelle les fichiers CSV seront copiés dans le Data Lake, on va lancer la conversion des fichiers en Parquet à une heure précise une fois que l'on est sûr qu'ils ont été copiés.

Pour programmer la conversion des fichiers, j'ai utilisé un Coordinator Ooozie qui lance les jobs Spark en charge de faire la conversion. En effet, pour une date donnée, un job Spark est lancé pour chaque type de fichier.

Partitionnement des données

Pour rendre la recherche plus performante, on a envisagé de partitionner les données par date. Par défaut, toutes les données sont stockées dans le même répertoire. Le partitionnement est une méthode permettant de séparer physiquement les données en se basant sur une ou plusieurs colonnes, cela rend les requêtes basées sur ces colonnes plus rapides. On obtient donc un répertoire par date.

Cette étape m'a pris un temps considérable, car j'avais testés plusieurs alternatives pour arriver finalement à une solution qui fonctionne, cela est du à la présence de quelques bugs dans Hive.

La solution pour laquelle j'ai opté consiste à créer une table Hive qui pointe vers un répertoire Parquet vide, tout en définissant le schéma des données. Pour stocker les données dans la table, le code consiste à les sauvegarder dans un premier temps dans une table temporaire, puis insérer le contenue de la table temporaire dans la table physique en spécifiant la valeur de la clé de partition.

L'organisation des fichiers sur le Lake ressemble à l'exemple suivant :

- Table1
    - As_of_date=20160707
    - As_of_date=20160708
    - As_of_date=20160711
    - As_of_date=20160712
    -
- Table2
- Table3
- …

As_of_date est le champ qui a servi de clé de partition.

Avancement du projet

Nous avons eu une première réunion avec les clients pour leur présenter ce qu'on envisage faire pour répondre à leur besoin, ce qui m'a permis de me mettre dans le contexte et de comprendre la vision du projet.

Ensuite, lorsque j'avais réussi à déployer le service qui leur permet de requêter les données, qui se limitaient à une seule table à ce moment là, nous avons fait une démonstration pendant laquelle je leur ai montré comment utiliser le service pour exécuter leurs requêtes. J'ai aussi préparé quelques requêtes simples, dans le but de tester le fonctionnement du service, je ne connaissais pas encore quelles étaient les requêtes qu'ils souhaitent faire tourner.

La prochaine étape, une fois que l'on a reçu le reste des tables qui sont au nombre de onze, consistait à intégrer les nouvelles données et les convertir en Parquet afin qu'elles soient exploitables.

Par la suite, j'ai reçu une requête de la part des clients, et en essayant de l'exécuter, je me suis aperçue qu'elle fait appel à d'autres tables que celles dont on dispose. Ce qui a déclenché une série de réunions avec les clients d'une part, et l'équipe qui se charge de nous fournir les données d'autre part, pour définir les tables qu'ils souhaitent que l'on intègre pour mieux répondre à leur besoins.

Nous attendons donc la réception de toutes les tables pour les intégrer pour pouvoir faire tourner des vraies requêtes provenant des clients.

Conclusion du chapitre

Le projet abordé dans ce chapitre constitue le sujet principal de mon stage. J'ai pu donc le suivre presque en autonomie. Pendant la réalisation de ce projet, sur lequel je continue à travailler jusqu'à la fin de mon stage, je suis amenée à prendre contact avec les clients, à faire des démonstrations pour leur montrer le fonctionnement de l'application, et à leur proposer un support quant à l'écriture de leurs requêtes. Le projet dépend aussi d'une autre équipe qui se charge de nous envoyer les données, je m'occupe donc aussi de les contacter que ce soit pour avancer sur la mise en place de l'envoi automatique, pour demander l'envoi manuel d'une table dont on a besoin, ou pour solliciter leur aide afin de comprendre le schéma des données.

Au jour où je rédige ce rapport, il me reste encore des bouts de code à développer pour automatiser le processus d'intégration des données et de leur conversion en Parquet.

CONCLUSION

Ce stage m'a permis de contribuer à la réalisation de tests de performance dans la cadre de la mise en place d'une nouvelle architecture basée sur les technologies Big Data. Les tests que j'ai réalisés n'ont pas donné lieu à des conclusions définitives, l'équipe continue ainsi à faire des tests, en faisant varier différents paramètres, en se basant sur les résultats intermédiaires des ces tests.

Cela m'a permis de m'intégrer au sein de l'équipe, de m'habituer au rythme de travail et d'assimiler les nouvelles connaissances fonctionnelles liées au métier de la finance d'une part, et les connaissances techniques liées technologies Big Data d'autre part.

Pendant ce stage, j'ai pu également suivre de la réalisation d'un projet en autonomie dont l'objectif consiste à mettre en place une application permettant de charger et requêter les données issues de certification et contrôle.

En travaillant sue ce projet, j'ai pu apprendre à écouter le client pour bien comprendre le besoin, ainsi qu'à bien agir quand le projet dépend de plusieurs équipes qui ont des disponibilités différentes.

Je souhaiterais aussi parler d'un point très important, à savoir la mise en production de certaines fonctionnalités de la nouvelle application. J'ai participé ainsi à toutes les réunions, et donc suivi toutes les étapes pour prendre connaissance des décisions qui ont été prises et des technologies qui ont été utilisées. J'ai pu donc me constituer une idée claire sur le passage du développement à la production en milieu professionnel.

En guise de conclusion, ce stage constitue pour moi une expérience professionnelle enrichissante et valorisante de point de vue technique et personnel. En effet, il s'agit de ma première expérience en entreprise, après avoir effectué mon stage de fin de 2ème année dans un laboratoire de recherche, où j'étais amenée à travailler au sein d'une équipe d'experts en Big Data, dans un métier loin de mes études, à savoir la banque et finance.

BIBLIOGRAPHIE

[i] WebHDFS REST API, https://hadoop.apache.org/docs/r1.0.4/webhdfs.html

[ii] Learning Spark, Holden Karau, Andy Konwinski, Patrick Wendell and Matei Zaharia, FévÉrier 2015, Première Édition, Copyright © 2015 Databricks. Tous droits réservés.

[iii] Apache Hive, https://cwiki.apache.org/confluence/display/Hive/Home

[iv] Apache Parquet, https://cwiki.apache.org/confluence/display/Hive/Parquet

[v] Apache HBase, http://hbase.apache.org/

[vi] Apache Oozie, http://oozie.apache.org/

[vii] Apache Knox, https://knox.apache.org/

[viii] Apache Kafka, http://kafka.apache.org/