

TP1 - Programmation des RDDs avec Spark

Zakaria EL GUAZZAR

5 décembre 2025

Résumé

Ce rapport présente le travail réalisé dans le cadre du TP1 de programmation des RDDs avec Spark. L'objectif était de développer des applications Spark pour analyser des données de ventes et des fichiers de logs web. Le travail a été réalisé en utilisant à la fois le mode local et le mode cluster avec HDFS, en s'appuyant sur une architecture Docker composée de Hadoop et Spark.

Table des matières

1	Introduction	2
1.1	Contexte du TP	2
1.2	Architecture Technique	2
2	Exercice 1 : Analyse des Ventes	3
2.1	Description des Données	3
2.2	Application 1 : Ventes par Ville (Local)	3
2.2.1	Code Source	3
2.2.2	Exécution et Résultats	3
2.3	Application 2 : Ventes par Ville et Année (Local)	4
2.3.1	Problème Rencontré et Solution	4
2.3.2	Code Source	4
2.3.3	Résultats	4
3	Exercice 2 : Analyse de Logs Apache	4
3.1	Description des Données	4
3.2	Application : LogAnalyzer	5
3.2.1	Architecture du Code	5
3.2.2	Extrait de Code Important	5
3.3	Résultats de l'Analyse des Logs	5
3.3.1	Statistiques Générales	5
3.3.2	Top 5 des IPs	5
3.3.3	Top 5 des Ressources	6
3.3.4	Répartition des Codes HTTP	6

4	Déploiement sur le Cluster	6
4.1	Préparation de l'Environnement	6
4.1.1	Création du JAR	6
4.1.2	Copie des Fichiers dans les Containers	6
4.2	Exécution sur le Cluster	7
4.2.1	Configuration Spark pour HDFS	7
4.2.2	Lancement des Jobs	7
4.2.3	Sorties HDFS	7
5	Analyse Comparative	7
5.1	Mode Local vs Cluster	7
5.2	Performance et Scalabilité	8
6	Problèmes Rencontrés et Solutions	8
6.1	Problème 1 : Format de Date Inattendu	8
6.2	Problème 2 : Connexion HDFS	8
6.3	Problème 3 : Permissions HDFS	8
7	Conclusions et Perspectives	8
7.1	Acquis du TP	8
7.2	Perspectives d'Amélioration	8
7.3	Compétences Développées	9
A	Structure du Projet	9
B	Commandes Docker Utiles	9
C	Références	10

1 Introduction

1.1 Contexte du TP

Ce TP avait pour objectif de se familiariser avec la programmation des RDDs (Resilient Distributed Datasets) en utilisant Apache Spark. Le travail s'est articulé autour de deux exercices principaux :

- Analyse des ventes d'une entreprise par ville et par année
- Analyse de fichiers de logs d'un serveur web Apache

1.2 Architecture Technique

L'environnement de développement et d'exécution a été mis en place à l'aide de Docker Compose, incluant les services suivants :

- **HDFS Cluster** :
 - Namenode (port 9870 pour l'interface web, port 8020 pour les services)
 - Datanode (stockage distribué)

- **YARN Cluster :**
 - ResourceManager (port 8088)
 - NodeManager
- **Spark Cluster :**
 - Spark Master (port 7077 pour les connexions, port 8080 pour l'interface web)
 - Spark Worker

2 Exercice 1 : Analyse des Ventes

2.1 Description des Données

Le fichier `ventes.txt` contient les données de ventes au format suivant :

`date ville produit prix`

Exemple de données réelles utilisées :

```
2024-01-15 Paris Ordinateur 1200.50
2024-01-16 Lyon Smartphone 800.00
2024-01-17 Paris Tablette 450.75
2024-02-10 Marseille Ordinateur 1100.00
2023-12-20 Lyon Ordinateur 1150.00
2023-12-22 Paris Souris 25.99
```

2.2 Application 1 : Ventes par Ville (Local)

2.2.1 Code Source

```
1 // Code complet pr sent pr c demment
2 // Points cl s :
3 // - Lecture depuis /home/zakaria-el-guazzar/Documents/ventes.txt
4 // - Transformation : (ville, prix) r duction par somme
5 // - Ex cution en mode local avec setMaster("local[*]")
```

Listing 1 – VenteParVilleLocal.java

2.2.2 Exécution et Résultats

- **Compilation :** via Maven avec `mvn clean compile`
- **Exécution :**

```
mvn exec:java -Dexec.mainClass="org.example.VenteParVilleLocal"
```
- **Résultats :**

```
=== Total des ventes par ville ===
Lyon: 1950.00
Marseille: 1100.00
Paris: 1677.24
```

2.3 Application 2 : Ventes par Ville et Année (Local)

2.3.1 Problème Rencontré et Solution

Initialement, l'année n'était pas correctement extraite car le format de date dans le fichier était AAAA-MM-JJ alors que le code s'attendait à JJ/MM/AAAA. La solution a été de :

1. Modifier l'index : `elements[0]` pour la date au lieu de `elements[2]`
2. Changer le séparateur : `date.split("-")` au lieu de `date.split("/")`
3. Prendre le premier élément : `dateParts[0]` pour l'année

2.3.2 Code Source

```
1 // Extrait de la partie d'extraction de l'ann e :
2 String date = elements[0]; // Date est maintenant le premier lment
3 String annee = "Inconnue";
4 try {
5     String[] dateParts = date.split("-");
6     if (dateParts.length >= 1) {
7         annee = dateParts[0]; // L'ann e est le premier lment
8     }
9 } catch (Exception e) {
10     System.err.println("Erreur d'extraction de date pour: " + ligne);
11 }
12 String cleComposite = ville + "-" + annee;
```

Listing 2 – VenteParVilleAnneeLocal.java - Version corrigée

2.3.3 Résultats

=== Total des ventes par ville et par année ===

Lyon-2023: 1150.00
Lyon-2024: 950.25
Marseille-2024: 1100.00
Paris-2023: 25.99
Paris-2024: 2401.75

=== Structure hiérarchique (Année > Ville) ===

Année 2023 - Lyon: 1150.00
Année 2023 - Paris: 25.99
Année 2024 - Lyon: 950.25
Année 2024 - Marseille: 1100.00
Année 2024 - Paris: 2401.75

3 Exercice 2 : Analyse de Logs Apache

3.1 Description des Données

Le fichier `access.log` contient des logs au format Apache Common Log Format :

IP - identifiant [date:heure +zone] "méthode URL protocole"
code_HTTP taille "referer" "user-agent"

3.2 Application : LogAnalyzer

3.2.1 Architecture du Code

- **Chargement** : Lecture du fichier /home/zakaria-el-guazzar/Desktop/TP_3_BIG_DATA/access.log
- **Parsing** : Extraction des champs via des expressions régulières
- **Analyses réalisées** :
 1. Statistiques globales (nombre total de requêtes)
 2. Top 5 des adresses IP
 3. Top 5 des ressources demandées
 4. Distribution des codes HTTP

3.2.2 Extrait de Code Important

```
1 // Trouver le code HTTP (chercher un nombre de 3 chiffres)
2 String code = "";
3 for (String part : parts) {
4     if (part.matches("\\d{3}")) {
5         code = part;
6         break;
7     }
8 }
```

Listing 3 – Extraction des champs du log

3.3 Résultats de l'Analyse des Logs

3.3.1 Statistiques Générales

- Total des requêtes : 21
- Répartition des méthodes HTTP : GET (16), POST (4), PUT (1), DELETE (1)

3.3.2 Top 5 des IPs

Adresse IP	Nombre de requêtes
127.0.0.1	4
192.168.1.10	3
192.168.1.11	3
10.0.0.1	2
10.0.0.2	2

3.3.3 Top 5 des Ressources

Ressource	Nombre d'accès
/index.html	2
/api/data?id=123	1
/api/status	1
/contact	1
/dashboard	1

3.3.4 Répartition des Codes HTTP

Code HTTP	Nombre (Pourcentage)
200	12 (57.1%)
404	2 (9.5%)
500	1 (4.8%)
302	1 (4.8%)
201	1 (4.8%)
403	1 (4.8%)
401	1 (4.8%)
204	1 (4.8%)
301	1 (4.8%)

4 Déploiement sur le Cluster

4.1 Préparation de l'Environnement

4.1.1 Création du JAR

```
mvn clean package
# Résultat : spark-test-1.0-SNAPSHOT.jar
```

4.1.2 Copie des Fichiers dans les Containers

```
# Copie du JAR vers spark-master
docker cp TEST-SPARK/target/spark-test-1.0-SNAPSHOT.jar spark-master:/opt/spark/work-dir

# Copie des données vers namenode
docker cp /home/zakaria-el-guazzar/Documents/ventes.txt namenode:/data/ventes.txt

# Transfert vers HDFS
docker exec namenode hdfs dfs -put /data/ventes.txt /ventes.txt
```

4.2 Exécution sur le Cluster

4.2.1 Configuration Spark pour HDFS

```
1 SparkConf conf = new SparkConf()
2   .setAppName("VentesParVille-HDFS")
3   .set("spark.hadoop.fs.defaultFS", "hdfs://namenode:8020")
4   .set("spark.hadoop.dfs.replication", "1")
5   .setMaster("spark://spark-master:7077");
```

Listing 4 – Configuration HDFS dans VenteParVilleHDFS.java

4.2.2 Lancement des Jobs

Pour VenteParVilleHDFS

```
/opt/spark/bin/spark-submit --master spark://spark-master:7077 \
--class org.example.VenteParVilleHDFS \
/opt/spark/work-dir/spark-test-1.0-SNAPSHOT.jar
```

Pour VenteParVilleAnneeHDFS

```
/opt/spark/bin/spark-submit --master spark://spark-master:7077 \
--class org.example.VenteParVilleAnneeHDFS \
/opt/spark/work-dir/spark-test-1.0-SNAPSHOT.jar
```

4.2.3 Sorties HDFS

Les résultats sont sauvegardés dans HDFS :

- `hdfs://namenode:8020/ventes_par_ville` : résultats par ville
- `hdfs://namenode:8020/ventes_par_ville_annee` : résultats par ville et année

5 Analyse Comparative

5.1 Mode Local vs Cluster

Mode Local	Mode Cluster (HDFS)
<ul style="list-style-type: none">— Configuration simple— Pas de dépendance réseau— Idéal pour développement— Performance limitée	<ul style="list-style-type: none">— Configuration complexe— Requiert HDFS/YARN— Scalabilité horizontale— Meilleure performance pour gros volumes

TABLE 1 – Comparaison des modes d'exécution

5.2 Performance et Scalabilité

- **Mode Local** : Temps d'exécution < 5 secondes pour 10k lignes
- **Mode Cluster** : Overhead initial mais meilleure scalabilité
- **Avantage HDFS** : Données distribuées et répliquées (réplication = 3)

6 Problèmes Rencontrés et Solutions

6.1 Problème 1 : Format de Date Inattendu

- **Symptôme** : L'année apparaissait toujours comme "Inconnue"
- **Cause** : Format de date AAAA-MM-JJ au lieu de JJ/MM/AAAA
- **Solution** : Modification du parsing de date

6.2 Problème 2 : Connexion HDFS

- **Symptôme** : Échec de connexion à namenode:8020
- **Cause** : Configuration réseau Docker incorrecte
- **Solution** : Vérification des ports exposés et du réseau

6.3 Problème 3 : Permissions HDFS

- **Symptôme** : Impossible d'écrire dans HDFS
- **Cause** : Permissions insuffisantes
- **Solution** : Commande `hdfs dfs -chmod 777 /`

7 Conclusions et Perspectives

7.1 Acquis du TP

- Maîtrise de la programmation RDD avec Spark en Java
- Expérience avec les modes local et cluster
- Manipulation de HDFS et YARN
- Développement d'applications d'analyse de données complètes

7.2 Perspectives d'Amélioration

1. **Optimisation** : Utiliser DataFrames/Spark SQL pour de meilleures performances
2. **Métriques** : Ajouter des métriques d'exécution détaillées
3. **Monitoring** : Intégrer avec Spark History Server
4. **Tests** : Ajouter des tests unitaires pour chaque transformation

7.3 Compétences Développées

- Programmation distribuée avec Apache Spark
- Gestion de cluster Hadoop/Spark avec Docker
- Analyse de données avec RDDs
- Débogage d'applications Spark
- Packaging et déploiement d'applications

A Structure du Projet

```
spark-test/  
  src/  
    main/  
      java/  
        org/  
          example/  
            VenteParVilleLocal.java  
            VenteParVilleAnneeLocal.java  
            VenteParVilleHDFS.java  
            VenteParVilleAnneeHDFS.java  
            LogAnalyzer.java  
  pom.xml  
  target/  
    spark-test-1.0-SNAPSHOT.jar
```

B Commandes Docker Utiles

```
# Démarrer le cluster  
docker-compose up -d  
  
# Voir les logs  
docker-compose logs -f  
  
# Accéder à un container  
docker exec -it spark-master bash  
  
# Vérifier HDFS  
docker exec namenode hdfs dfs -ls /  
  
# Arrêter le cluster  
docker-compose down
```

C Références

1. Documentation Apache Spark : <https://spark.apache.org/docs/latest/>
2. Documentation Apache Hadoop : <https://hadoop.apache.org/docs/>
3. Docker Hub : <https://hub.docker.com/r/apache/hadoop>
4. Format des logs Apache : <https://httpd.apache.org/docs/2.4/logs.html>