TP Spark

Saison 2022-2023

Composante : ESIR

Spécialité : Technologies de l'information option informatique 3ème année

Module : Data management for big data

Chargée de TP : Maria Massri, maria.massri@irisa.fr

Contexte

Ce TP est à faire en monôme ou binôme. La documentation nécessaire est celle de **Spark** [1] et la libraire **GraphX** [2]. Pour cette partie du TP, pas de compte rendu n'est demandé. Cependant, nous vous demandons de déposer votre code sur **Git** et de partager le lien du projet avec votre chargé de TP.

1 Pré-requis

Nous conseillons d'utiliser l'IDE IntelliJ IDEA pour ce TP.

1. Créez un projet Scala (Vous trouverez les étapes dans Intellij sur `https://docs.scala-lang.org/getting-started/intellij-track/building-a-scala-project-with-intellij-and-sbt.html`). ATTENTION : Sélectionner la version Scala 2.13.10.
2. Récupérez l'archive de Java 11 sur `https://openjdk.java.net/install/`, si vous ne l'avez pas déjà installé sur vos machines et configurez l'IDE avec cette version.
3. Enfin, ajoutez les dépendances Spark-core et Spark-graphx (version 3.3.1) au fichier **.sbt**.

2 Préparation du jeu de données

Le jeux de données utilisé dans ce TP provient de *PlayerUnknown's Battlegrounds* [3], un jeu vidéo de tir en ligne de type *Battle Royale* dont le but est d'être le dernier joueur en vie, en équipe ou en solo. Téléchargez le jeux de données disponible sur Kaggle [4] (Fichier $agg\_match\_stats\_0.csv$). Le jeu de données contient ce que les actions des joueurs pour chaque partie (classement, distance parcourue, nombre d'éliminations, etc.).

1. Décompressez le jeu de données dans un espace de travail.
2. Combien de lignes fait ce fichier ? (voir la commande `wc`).
3. Nous travaillerons d'abord sur un échantillon contenant les 100 mille premières lignes, générez celui-ci. (voir la commande `head`).

1. `https://spark.apache.org/docs/latest/rdd-programming-guide.html`
2. `https://spark.apache.org/docs/latest/graphx-programming-guide.html`
3. `https://fr.wikipedia.org/wiki/PlayerUnknown\%27s\_Battlegrounds`
4. `https://www.kaggle.com/skihikingkevin/pubg-match-deaths`

3 Les meilleurs joueurs

L'objectif est d'être le dernier en vie, mais certains joueurs soutiennent qu'il est nécessaire d'éliminer un maximum de concurrents. Nous allons vérifier cette affirmation en comparant les joueurs selon ces deux conditions, à vous de choisir **celle que vous souhaitez explorer**. L'attribut `player_kills` donne le nombre d'éliminations et `team_placement` la position en fin de partie.

1. Chargez le jeu de données. (voir `textFile`)
2. Pour chaque partie, obtenez uniquement le nom du joueur et son nombre d'éliminations **ou** sa position. (voir `map`)
3. Obtenez la moyenne des éliminations **ou** de la position de chaque joueur, ainsi que le nombre de parties concernées. (voir `reduceByKey` ou `groupByKey`)
4. Obtenez les 10 meilleurs joueurs selon les éliminations **ou** la position. (voir `sortBy`)
5. Certains joueurs n'ayant joué qu'une partie, nous souhaitions ne garder que ceux ayant au moins 4 parties. (voir `filter`)
6. Si vous observez un joueur particulier, traitez-le de la manière appropriée.
7. En partageant avec vos camarades qui ont exploré l'autre condition, donnez votre avis sur l'affirmation de départ.

4 Score des joueurs

Nous allons assigner un score à chaque joueur selon ses actions lors de chaque partie. Chaque joueur gagnerait 50 points par assistance, 1 point par dommage causé, 100 points par élimination et 1000 points s'il finis à la première place, 990 à la deuxième, ainsi de suite.

1. Développez une fonction spécifique pour obtenir le score, et obtenez les 10 meilleurs joueurs selon ce critère.
2. Comparez ce classement avec les deux précédents critères.

5 Persistance (*Bonus*)

Dans cette partie, nous travaillerons sur le jeu de données de 2Go. Vous remarquerez qu'on accède souvent au même état d'un jeu de données pour différents traitements. Spark permet de mettre un état donné en cache afin de ne pas avoir à charger les données à chaque action.

1. Obtenez, en plus des meilleurs joueurs, le nombre total de joueurs distincts. (voir count, Distinct)
2. A partir de la documentation du mécanisme de persistance de Spark 3, choisissez un mode de persistance à utiliser et justifiez ce choix.
3. Appliquez la persistance sur l'état du jeu de données qui vous semble le plus opportun pour répondre à la première question ci-dessus. (voir persist)
4. Obtenez les meilleurs joueurs et le nombre de joueurs en mesurant le temps de calcul de ces deux opérations avec et sans persistance sur 3 exécutions.
5. Observez-vous un gain de performance ? Comment l'expliquez-vous ?