M2 INFO - Données Distribuées - TP Spark

Nampoina Andriamilanto

Saison 2020-2021

Responsable UE : David Gross-Amblard

Enseignants : Nampoina Andriamilanto, David Gross-Amblard

Contact : `tompoariniaina.andriamilanto@irisa.fr`, `david.gross-amblard@irisa.fr`

Contexte

Ce TP sur Spark est à faire en monôme ou binôme. La documentation nécessaire est celle de **Spark SQL** sur `https://spark.apache.org/docs/latest/sql-programming-guide.html`.

1 Installation de Spark

1. Récupérez l'archive de la version la plus récente de Spark (3.0.1) sur `https://spark.apache.org/downloads.html`.
2. Décompressez-là dans un espace de travail.
3. Testez l'installation en exécutant un interpréteur ou via l'interface web sur `http://localhost:4040`.

Vous pouvez utiliser l'interpréteur Python3 (`[SPARK_HOME]/bin/pyspark`) ou bien l'interpréteur Scala (`[SPARK_HOME]/bin/spark-shell`), dans ce cas pensez à **noter les instructions réalisées**. Nous conseillons d'utiliser des scripts dont des exemples sont présents dans `[SPARK_HOME]/examples/src/main`.

2 Préparation du jeu de données

Le jeu de données vous est fournis sous forme d'une archive nommée `agg_match_stats_0.csv.zip`, aussi disponible sur FileSender [1]. Ces données proviennent de *PlayerUnknown's Battlegrounds* [2], 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. Seul un des dix fichiers du jeu de données complet vous est fournis, la version complète et sa description est disponible sur Kaggle [3]. 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://filesender.renater.fr/?s=download&token=ecee01a7-dfd4-421c-a78f-bcd7f2c5f788`
2. `https://fr.wikipedia.org/wiki/PlayerUnknown%27s_Battlegrounds`
3. `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 `spark.read`)
2. Pour chaque partie, obtenez uniquement le nom du joueur et son nombre d'éliminations **ou** sa position. (voir `select`)
3. Obtenez la moyenne des éliminations **ou** de la position de chaque joueur, ainsi que le nombre de parties concernées. (voir `groupBy`, `agg`)
4. Obtenez les 10 meilleurs joueurs selon les éliminations **ou** la position. (voir `orderBy`)
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 Persistance

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 `countDistinct`)
2. A partir de la documentation du mécanisme de persistance de Spark [4], 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 ?

5 Exécution distribuée (optionnel)

Nous allons maintenant passer à lexécution distribuée. Avant toute exécution distribuée, vous devez **faire vérifier vos instructions** par le chargé de TP. Des instructions non optimisées peuvent vous faire perdre de précieuses minutes !

Pour l'exécution distribuée, formez un groupe avec d'autres binômes au sein duquel chaque binôme prendra le rôle du `master` à tour de rôle. Mesurez le temps d'exécution de vos instructions (voir commande shell `time` ou bibliothèque Python `timeit`). Il est **vivement conseillé d'utiliser un script**.

5.1 Lancement du Master

Cette partie ne concerne que le `master` courant.

1. Sur le master, exécutez `[SPARK_HOME]/sbin/start-master.sh -h [MASTER_IP] -p 7077 -webui-port 8080`.

4. `https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-persistence`

2. Vérifiez que le master est bien lancé en accédant à `http://[MASTER_IP]:8080`.
3. Vérifiez que les workers soient bien liés après leur lancement.
4. Pensez à fermer le master lorsque vous n'aurez plus ce rôle avec `[SPARK_HOME]/sbin/stop-master.sh`.

5.2 Lancement d'un Worker

1. Créez un lien symbolique dans `/tmp/dd-datas` vers un dossier contenant les données.
2. Exécutez la commande ci-dessous, en configurant la même quantité de CPU et de mémoire allouée aux workers entre les différentes machines (suggestion : 2 curs et 1G de mémoire).
3. `[SPARK_HOME]/sbin/start-slave.sh -c [NB_CORES] -m [MEMORY] -h [WORKER_IP] -webui-port 8081 spark://[MASTER_IP]:7077`

5.3 Fonctionnement du mode distribué

— Lors de l'exécution distribuée, vos opérations seront transmises au `master`, celui-ci les sérialise et les transmet à son tour aux différents `workers`.
— Les `workers` devront avoir accès aux données au travers d'un **même chemin**, d'où la nécessité de créer un lien symbolique dans `/tmp/dd-datas`.
— Les `workers` transmettent les résultats au master, qui lui les transmettra à son tour au programme l'ayant exécuté. Si vous lancez un `collect` de taille importante, l'entièreté des données est envoyée et peut provoquer une surcharge !

5.4 Exécution distribuée

1. Configurez dans votre script la **connexion au master** (par exemple en initialisant le `SparkContext` via `SparkContext("spark://[MASTER_IP]:7077")`).
2. Changez le chemin d'accès aux fichiers en `/tmp/dd-datas`.
3. Mesurez vos temps d'exécution sur **au moins 3 exécutions**.
4. Ouvrez les moniteurs de ressource des différentes machines pour observer la charge de chacune d'entre elles.
5. Comparez les temps d'exécution avec les autres binômes de votre groupe, et identifiez les raisons probables des différences qui pourraient survenir.

6 Avant de partir

1. Vérifiez que vous avez bien pris note des instructions et résultats de la séance.
2. Fermez le contexte Spark s'il est ouvert via `[SparkContext].stop()`.
3. Fermez le *master* et/ou le *worker* qui tournerait sur votre machine via `[SPARK_HOME]/sbin/stop-slave.sh` et/ou `[SPARK_HOME]/sbin/stop-master.sh`.
4. Nettoyez les fichiers temporaires de travail générés dans `[SPARK_HOME]/work`.