TP Spark

Ce TP est à effectuer en binôme (ou trinôme) possédant a minima une machine. Il durera **4 séances de 2 heures**, et la notation se fera en fonction des différents points de contrôle auxquels vous serez rendus.

Même si aucun compte-rendu n'est demandé à la fin, vous devrez présenter vos résultats et les instructions réalisées. Pensez donc à **noter vos instructions** (idéalement, utilisez un script) et vos **réponses aux questions** ! Une partie bonus est disponible pour les plus rapides.

Toute la documentation nécessaire à ce TP est disponible sur la documentation des RDDs, ainsi que celle de l'API pyspark. Enfin, dans `[SPARK_HOME]/python/docs` vous pouvez générer la documentation de pyspark (le *wrapper* python de Spark) via la commande `make html` . Les fichiers générés se trouvent ensuite dans `[SPARK_HOME]/python/docs/_build/html` .

1 Mise en route

1.1 Installation de Spark

Récupérez l'archive de la **version 2.4.4** de Spark indiquée avec *Pre-built for Apache Hadoop 2.7* sur https://spark.apache.org/downloads.html. Décompressez-là dans un espace de travail désigné.

Pour manipuler avec Spark, vous pouvez utiliser les **interpréteurs** Python3 `[SPARK_HOME]/bin/pyspark` ou Scala `[SPARK_HOME]/bin/spark-shell` , dans ces cas pensez à **noter les instructions réalisées**. Il est conseillé d'utiliser des **scripts** en Python3, Scala ou Java, dont des exemples sont disponibles dans `[SPARK_HOME]/examples/src/main` .

Pour **tester l'installation**, exécutez l'interpréteur de votre choix et vérifiez que celui-ci arrive à l'étape vous demandant des instructions. Vous pouvez aussi vérifier que l'interface graphique est accessible au travers d'un navigateur sur http://localhost:4040.

1.2 Préparation du jeu de données

Le jeu de données vous sera fourni sous forme d'une archive nommée `agg_match_stats_0.csv.zip` que vous décompresserez dans votre répertoire de travail. Ces données proviennent de

PlayerUnknown's Battlegrounds (PUBG pour les intimes), un jeu vidéo de tir en ligne de type *Battle Royale* dont l'objectif est d'être le dernier joueur en vie, en équipe ou en solo.

Description

Seul un des 10 fichiers du jeu de données complet vous est fournis, la version complète ainsi qu'une description des fichiers est disponible sur Kaggle (compte requis pour télécharger).

Le fichier `agg_match_stats_0.csv` concerne les parties, avec des informations sur ce qu'a pu y faire chaque joueur (classement, distance parcourue, nombre d'éliminations, etc.). Notez que la **première ligne contient l'en-tête** décrivant les différents champs.

Échantillonnage

Lorsqu'on traite une large quantité de données, il est souvent nécessaire de développer et tester ses scripts sur un échantillon.

1. Combien de lignes fait ce fichier ? (voir la commande `wc`).
2. Nous allons travailler sur un échantillon commun, disons les **100 mille premières lignes** (voir la commande `head`).

2 Traitements simples

Le but du jeu est d'être le dernier en vie, mais certains joueurs persistent à dire que ça ne se peut qu'en éliminant un maximum d'ennemis. Nous allons vérifier cette affirmation en comparant les meilleurs joueurs selon ces deux conditions, à vous de choisir **celle que vous souhaitez explorer**. Le nombre d'élimination correspond à la colonne nommée `player_kills`, le placement à la colonne `team_placement`, et une ligne du jeu de données correspond à une partie effectuée par un joueur.

3. Pour chaque ligne de donnée, obtenez uniquement le nom du joueur et son nombre d'élimination **ou** son placement. (voir `textFile`, `map`)
4. Obtenez la somme des éliminations **ou** des placements de chaque joueur. (voir `reduceByKey`)
5. Obtenez le nombre d'élimination **ou** le placement moyen de chaque joueur, ainsi que le nombre de parties. (voir `reduceByKey` ou `groupByKey`)
6. Obtenez les 10 meilleurs joueurs selon le nombre d'élimination **ou** le placement. (voir `sortBy`)
7. Certains joueurs bien classés n'ayant joué qu'une partie, nous souhaitions ne garder que ceux ayant au moins 4 parties. (voir `filter`)
8. Si vous croisez un joueur particulier dans ce classement, traitez-le de la manière appropriée. (voir `filter`)

9. Finalement, donnez votre avis sur l'affirmation de départ.

Avant de partir

- Vérifiez que vous avez bien pris note des instructions et résultats de la séance.
- Fermez votre contexte Spark via `sc.stop()` .

Félicitation, vous avez fini la première partie. Prévenez le chargé de TP !

3 Traitements avancés

Persistance

Lors de l'analyse d'un jeu de données, on accède souvent à un même RDD. Spark permet de sauvegarder celui-ci afin de ne pas avoir à charger les données à chaque fois.

10. Obtenez, en plus des meilleurs joueurs, le nombre total de joueurs. (voir `count` , `distinct`)
11. Utilisez le mécanisme de persistance offert par Spark selon le mode de votre choix, justifiez celui-ci. (voir `persist`)
12. Obtenez les meilleurs joueurs et le nombre de joueurs, et mesurez le temps de calcul de ces **deux opérations** avec et sans persistance sur **3 exécutions**.
13. Quel gain de performance obtenez-vous ?

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.

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

Accumulateurs

Au passage, nous souhaitons aussi compter le nombre de fois qu'un joueur a au moins 5 fois plus d'assistances que d'éliminations. Au lieu de développer des instructions particulières, on peut le faire **durant le calcul du score** via un accumulateur, nous évitant ainsi de reparcourir les données.

16. Obtenez le nombre de fois qu'un joueur a 5 fois plus d'assistance que d'élimination grâce à un accumulateur. (voir `accumulator`)

Variables partagées

La fonction de score assigne des poids arbitraires aux actions du joueur, il serait alors intéressant de les saisir dynamiquement. Pour se faire, on peut passer par une variable partagée.

17. Récupérez dynamiquement les poids à associer à chaque information.
18. Utilisez une variable partagée pour évaluer le score de manière dynamique. (voir `broadcast`)

Avant de partir

- Vérifiez que vous avez bien pris note des instructions et résultats de la séance.
- Nettoyez vos RDDs persistés en mémoire avec `[votre RDD].unpersist()`.
- Fermez votre contexte Spark via `sc.stop()`.

Félicitation, vous avez fini la deuxième partie. Prévenez le chargé de TP !

4 Exécution distribuée

Nous allons maintenant passer à l'exécution distribuée. Avant toute exécution distribuée, vous devez **faire vérifier vos instructions** par le chargé de TP. Une petite erreur peut 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 commande `time` ou bibliothèque Python `timeit`). Il est **vivement conseillé d'utiliser un script**.

4.1 Mise en route

Lancement du Master

Cette partie ne concerne que le *master* courant.

- Sur le master, exécutez `[SPARK_HOME]/sbin/start-master.sh -h [MASTER_IP] -p 7077 --webui-port 8080`.
- Vérifiez que le master est bien lancé en accédant à `http://[MASTER_IP]:8080`.

- Vérifiez que les workers soient bien liés après leur lancement.
- Pensez à fermer le master lorsque vous n'aurez plus ce rôle avec `[SPARK_HOME]/sbin/stop-master.sh`.

Lancement d'un Worker

- Créez un lien symbolique dans `/tmp/dd-datas` vers un dossier contenant les données.
- Exécutez la commande ci-dessous, en configurant la même quantité de CPU et de mémoire allouée aux workers entre les différentes machines (suggestion : 2 coeurs et 1G de mémoire).

```
[SPARK_HOME]/sbin/start-slave.sh -c [NB_CORES] -m [MEMORY] -h [WORKER_IP] --webui-port 8081
spark://[MASTER_IP]:7077
```

Fonctionnement du mode distribué

- Lors de l'exécution distribuée, vos opérations seront transmis 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 !

4.3 Exécution distribuée

Vous pouvez maintenant exécuter votre script ou suite d'instruction, et pouvoir répondre aux questions précédentes, cette fois-ci sur le **fichier entier de 2Go**.

19. Configurez dans votre script la **connexion au master** comme suit `sc = SparkContext("spark://[MASTER_IP]:7077")`. Si vous êtes en mode intéractif, pensez à **fermer le précédent contexte** avec `sc.stop()`.
20. Changez le chemin d'accès aux fichiers en `/tmp/dd-datas`.
21. Mesurez vos temps d'exécution sur **au moins 3 exécutions**.
22. Ouvrez les moniteurs de ressource des différentes machines pour observer la charge de chacune d'entre-elles.
23. Comparez les temps d'exécution avec les autres binômes de votre groupe, et identifiez les raisons probables des différences qui pourraient survenir.

Avant de partir

- Vérifiez que vous avez bien pris note des instructions et résultats de la séance.
- Nettoyez vos RDDs chargés en mémoire avec `[votre RDD].unpersist()`.
- Fermez votre contexte Spark via `sc.stop()`.
- Fermez le *master* et/ou le *worker* qui tourne sur votre machine via `[SPARK_HOME]/sbin/stop-slave.sh` et/ou `[SPARK_HOME]/sbin/stop-master.sh`.
- Nettoyez les fichiers temporaires de travail générés dans `[SPARK_HOME]/work`.

Félicitation, vous avez fini la troisième partie. Prévenez le chargé de TP !

5 Bonus

Cette partie contient les exercices bonus pour ceux qui auraient de l'avance. Ici, peu d'indications vous seront donné, n'hésitez pas à discuter de vos solutions avec le chargé de TP. Même si vous n'avez pas obtenu les résultats finaux, la présentation de vos résultats/instructions intermédiaires sera pris en compte dans la notation.

5.1 Mon cher nemesis (niveau moyen)

Certains joueurs ont tendance à se croiser et à s'entre-éliminer. Retrouvez donc les pires ennemis, soit les joueurs qui se sont le plus entre-éliminés.

5.2 Apprendre des meilleurs (niveau difficile)

Pour se perfectionner à un jeu, mieux vaut apprendre des meilleurs ! Il serait alors intéressant de retrouver les armes les plus utilisées par les meilleurs joueurs, et celles utilisées par les pires, afin de voir si certaines semblent plus efficaces.

5.3 Genki Dama (niveau facile)

Si vous avez suffisamment de temps, vous pouvez avoir les résultats du jeu de données complet qui fait environ 10Go. Pour ce faire, vous aurez besoin d'aide ! Sélectionnez votre champion (le meilleur script) et demandez à vos camarades de vous prêter main forte (et surtout leur machine).

Comparez alors les temps d'exécution en fonction du nombre de machines, si possible effectuez au moins 2 exécutions pour chaque essais.

5.4 Proposition de bonus (niveau ???)

Si vous avez une question à laquelle le jeu de données peut apporter une réponse, n'hésitez pas à proposer celle-ci comme bonus au chargé de TP. Il pourra vous donner une estimation de son niveau de difficulté, et valider celle-ci. Dans tous les cas, faites valider votre proposition avant de vous lancer.

Andriamilanto Tompoariniaina (tompoariniaina.andriamilanto@irisa.fr) 2019 - 2020