Projet de Programmation Large Echelle

Pablo Tomas
Erwan Dupland

16 janvier 2020

Table des matières

1 Présentation **3**

2 Rapport **4**
2.1 Question 1 4
2.1.a 4
2.1.b 4
2.1.c 4
2.2 Question 2 5
2.3 Question 3 5
2.4 Question 4 5
2.4.a 6
2.4.b 6
2.5 Question 5 6
2.6 Question 6 6
2.6.a 7
2.6.b 7
2.7 Question 7 7
2.8 Question 8 7

3 Résultats obtenus **7**
3.1 Question 1 8
3.1.a 8
3.1.b 8
3.1.c 9
3.2 Question 2 11
3.3 Question 3 11
3.4 Question 4 12
3.4.a 12
3.4.b 12
3.5 Question 5 12
3.6 Question 6 13
3.6.a 13
3.6.b 13
3.7 Question 7 14

4 Bilan **14**

1 Présentation

Pour des raisons de lisibilité et d'efficacité, il a été decidé de créer deux applications. La première application est une application MapReduce. Le but de cette application est de parser le fichier *phases.csv* pour le transformer sous forme de **Sequence File**. Appelons la donc **PhaseToSequenceFile**. Sous la forme de SequenceFile, les données seront traitées plus efficacement dans la deuxième application. La seconde application est une application Spark qui est destiné à répondre aux questions posées par le sujet. Appelons cette application **PhaseSpark**.

Le projet est réparti sous la forme de 19 fichiers. Le fichier **PhaseSequenceFile.java** est celui qui contient les instructions de l'application **PhaseToSequenceFile**. En réalité il ne s'agit pas d'une application MapReduce puisque le job n'utilise aucun Reducer. Il n'utilise qu'un Mapper qui parsera chaque Record du fichier pour le transformer en un objet **PhaseWritable** qui sera stocké dans les Sequence Files intermédiaires. Le fichier **ProjetSpark.java** est celui qui contient les instructions de l'application **PhaseSpark**.

Les classes **Question1** jusqu'à **Question7** représentent les réponses apportées aux différentes questions du sujet. Les classes **QuestionInt**, **QuestionLong**, **QuestionTopK** et **QuestionTotalTime** sont les classes mères de certaines classes Question. Ces classes servent principalement à factoriser du code utilisé dans leurs classes filles. Les classes **Distribution**, **TotalTime**, **PercentTime**, **Hours** et **TopK** permettent de stocker les données interessantes pour répondre aux différentes questions du sujet.

Pour compiler le projet avec Maven, il n'était pas évident de séparer les deux applications dans deux projets distincts avec chacun un fichier **pom.xml** pour les compiler separemment. Le principal problème de cette méthode était que les deux jobs avaient besoin de la classe **PhaseWritable**. Plutôt que d'avoir ce fichier présent dans chaque projet, il a été decidé de mettre les fichiers des deux applications dans un même dossier et de modifier le fichier **pom.xml** à chaque fois qu'il était necessaire de compiler une application. Pour cela le fichier **pom.xml** contient une partie commentée et une partie qui ne l'est pas. La partie commentée correspond au fichier **pom.xml** de l'application **PhaseToSequenceFile**. Il suffit de la décommenter et de commenter l'autre pour lancer la compilation de la première application.

Pour lancer l'application **PhaseToSequenceFile** il faut utiliser Maven (après avoir effectué les modifications décrites dans le paragraphes précedent) en se positionnant là où se trouve le fichier **pom.xml** (c'est à dire à la racine du projet). Il faut ensuite lancer la commande suivante où il faut préciser l'argument $input\_dir$ qui correspond au fichier contenant les données que l'on va traiter pendant le projet et l'argument $output\_dir$ qui correspond au dossier où seront stockés les Sequence files retournés par l'application :

```
yarn jar ./target/mapreduce_maven-0.0.1.jar intput_dir output_dir
```

Pour lancer l'application **PhaseSpark**, il faut à nouveau effectuer des modifications décrites dans le paragraphe dédié au fichier **pom.xml**. Il faut ensuite lancer la commande suivante où il faut préciser l'argument $input\_dir$, qui correspond au dossier où se trouve les sequence files créés par la première application et l'argument $output\_dir$ qui correspond au dossier où seront stockées les réponses aux questions du sujet :

```
spark-submit --num-executors X --executor-cores X --executor-memory XXXM
    --master yarn --class bigdata.ProjetSpark target/spark_maven-0.0.1.
    jar intput_dir output_dir
```

2 Rapport

2.1 Question 1

2.1.a .....

Pour récuperer des informations sur la durée des phases *non-idle*, il a fallu, dans un premier temps, effectuer un filtre sur le **JavaRDD** stockant toutes les données. Sur ce **JavaRDD** filtré il ne reste plus qu'à créer un **JavaDoubleRDD** sur la durée des phases à l'aide de la fonction $mapToDouble()$. Cette classe est dotée d'une fonction $histogram()$ qui permet de construire un histogramme à partir d'un tableau passé en paramètre définissant les intervalles de l'histogramme.

A partir de la classe **JavaDoubleRDD** il est possible de créer un objet de la classe **StatCounter** qui a tout un ensemble de fonction permettant de réaliser des opérations statistiques sur les données qui nous intéressent. Les fonctions qui nous intéressent ici sont : $min()$, $max()$ et $mean()$. Pour trouver la médiane, le premier et le troisième quartile, il a fallu isoler les durées des phases *non-idle* grâce à la fonction $map()$. Le **JavaRDD** obtenu a ensuite été trié et complété en un **JavaPairRDD** pour que chaque durée soit retrouvée avec un index. Cela a été possible grâce aux fonctions $sortBy()$ et $zipWithIndex()$. Il ne reste plus qu'à utiliser la fonction $lookup()$ pour retrouver les durées aux index de la médianes et des quartiles.

2.1.b .....

Pour récupérer des informations sur la distribution de la durée des phases *idle*, il a fallu effectuer les mêmes opérations que pour la question 1.a.

2.1.c .....

Pour récupérer des informations sur la distribution de la durée des phases où chaque motif apparaît seul, nous avons utilisé une boucle et un **JavaRDD** temporaire. Dans cette boucle, le **JavaRDD** temporaire est modifié à chaque tour de boucle grâce à la fonction $filter()$. La fonction $filter()$ prend en paramètre une $lambdafunction$. Sachant que

nous devions mettre à jour la fonction de filtre à chaque tour de boucle (pour chaque motif d'accès) et qu'une $lambdafunction$ ne peut pas utiliser une variable non $final$ défini en dehors de sa définition, il a fallu utiliser un **AtomicInteger** pour répondre à ce besoin.

De cette manière il nous été possible de modifier la $lambdafunction$ à chaque tour de boucle. Une fois le **JavaRDD** trié pour un certain motif d'accès, il ne reste plus qu'à recupérer les données de la disctribution de la durée de la même manière que nous l'avons décrit dans la question 1.a. Le programme passe ensuite au motif d'accès suivant.

2.2 Question 2

Pour récupérer des données sur la distribution du nombre de motifs d'accès par phases *non-idle*, il a fallu effectuer les mêmes opérations que pour la question 1.a. Il faut cependant isoler le nombre de motifs d'accès par phase plutôt que leur durée (lors de la création du **JavaDoubleRDD** grâce à la fonction $mapToPair()$ et lors de la recherche de la médiane, du premier et du troisième quartile grâce à la fonction $map()$.

2.3 Question 3

Pour récupérer des données sur la distribution du nombre de jobs par phases *non-idle*, il a fallu effectuer les mêmes opérations que pour la question 1.a. Il faut cependant isoler le nombre de jobs par phase plutôt que leur durée (lors de la création du **JavaDoubleRDD** grâce à la fonction $mapToPair()$ et lors de la recherche de la médiane, du premier et du troisième quartile grâce à la fonction $map()$.

2.4 Question 4

Pour répondre à la question suivante, nous avons réinterprété un point du sujet : Si sur une phase, plusieurs jobs ont accédé au PFS, combien de temps a pris chaque job puisque c'est la durée totale de la phase qui est precisée ? L'énoncé du projet n'apportait aucun élément de réponse à ce sujet. Nous avons decidé de prendre en compte le pire cas : un job peut prendre la durée totale de la phase à lui seul. Le temps total de chaque job de la phase est donc incrémenté par la durée totale de la phase. C'est donc le temps total d'accès au système dans le pire des cas par job qui est recherché dans cette question.

Pour récupérer les temps totaux d'accès au système par job, il a fallu d'abord récupérer les phases *non-idle*. Ensuite à partir de ce **JavaRDD**, il a fallu le modifier de telle sorte à obtenir un **JavaPairRDD** dont les clés sont les jobs de la phase et les valeurs sont la durée de la phase. On réalise cela grâce à la fonction $mapToPair()$. On effectue ensuite une operation sur le **JavaPairRDD** pour que chacune de ses clés soit parsé et qu'une clé ne représente plus qu'un job tout en gardant la même valeur. La fonction $flatMapToPair$ permet d'obtenir ce résultat. On finit par utiliser la fonction $reduceByKey()$ pour sommer les valeurs qui ont la même clé. Ainsi on obtient un **JavaPairRDD** avec des clés uniques et qui représentent chaque jobs et dont la valeur est le temps total d'accès au système.

2.4.a .....

Pour récupérer des données sur la distribution du temps total d'accès au système par job, il a fallu travailler sur le **JavaPairRDD** décrit dans la section au dessus. On effectue ensuite les mêmes opérations que pour la question 1.a.

2.4.b .....

Pour récupérer le Top 10 des jobs en temps total d'accès au système, il faut travailler à partir du **JavaPairRDD** défini dans la section de la question 4. C'est dans la classe abstraite **QuestionTopK** que l'on s'occupe de trouver le Top 10 d'un **JavaPairRDD**. Dans cette classe on va définir une *inner-class* **InnerComparator** qui implemente les interfaces **Comparator** et **Serializable**. Cette *inner-class* va être utilisée dans la fonction $top()$ de notre **JavaPairRDD** et nous ressortir le Top 10 sous forme de **List**.

2.5 Question 5

Pour récupérer le temps total d'accès au système par les phases $idle$. On filtre d'abord les phases $idle$ du **JavaRDD** contenant toutes les données. Ensuite on utilise la fonction $aggregate()$ sur celui-ci. Pour utiliser la fonction $aggregate()$, notre classe **TotalTime** implémente l'interface **Serializable** afin qu'on puisse l'utiliser comme paramètre pour la fonction.

2.6 Question 6

Pour répondre à la question suivante, nous avons réinterprété un point du sujet : Si sur une phase, plusieurs motif d'accès ont accédé au PFS, combien de temps a pris chaque motif d'accès puisque c'est la durée totale de la phase qui est precisée ? L'énoncé du projet n'apportait aucun élément de réponse à ce sujet. Nous avons decidé de prendre en compte le pire cas : un motif d'accès peut prendre la durée totale de la phase à lui seul. Le temps total de chaque motif d'accès de la phase est donc incrémenté par la durée totale de la phase. C'est donc le temps total d'accès au système dans le pire des cas par motif d'accès qui est recherché dans cette question.

Pour récupérer les temps totaux d'accès au système par motif d'accès, il a fallu d'abord récupérer les phases *non-idle*. Ensuite à partir de ce **JavaRDD**, il a fallu le modifier de telle sorte à obtenir un **JavaPairRDD** dont les clés sont les patterns utilisés lors de la phase et les valeurs sont la durée de la phase. On réalise cela grâce à la fonction $mapToPair()$. On effectue ensuite une operation sur le **JavaPairRDD** pour que chacune de ses clés soit parsé et qu'une clé ne représente plus qu'un motif d'accès tout en gardant la même valeur. On ajoute un "S" devant la clé du motif d'accès pour signifier que le motif d'accès n'a pas été utilisé seul lors de la phase. La fonction $flatMapToPair$ permet d'obtenir ce résultat. On finit par utiliser la fonction $reduceByKey()$ pour sommer les valeurs qui ont la même clé. Ainsi on obtient un **JavaPairRDD** avec des clés uniques et qui représentent chaque motifs d'accès seul et en simultané et dont la valeur est le temps total d'accès au système.

2.6.a .....

En plus du **JavaPairRDD** décrit dans la section précedente, on va récupérer le temps total des phases *non-idle* de la même manière que celle décrite dans la section de la question 5. Cette donnée supplémentaire va nous permettre de réaliser des pourcentages. On va utiliser la fonction $collect()$ sur le **JavaPairRDD** pour avoir une représentation sous forme de liste de nos données. Comme il n'y a que 22 patterns, il n'y aura que 44 entrées dans la liste construite grâce à la fonction $collect()$. Tant qu'il y a peu de valeur dans le **JavaPairRDD**, ce n'est pas dangereux d'utiliser cette fonctionnalité.

2.6.b .....

Pour récupérer le Top 10 des motifs d'accès en temps total d'accès au système, il faut travailler à partir du **JavaPairRDD** defini dans la section de la question 6. On va ensuite effectuer les mêmes opérations que celles décrites dans la section de question 4.a. pour récuperer le top 10.

2.7 Question 7

Pour récupérer toutes les plages horaires comportant 4 motifs d'accès rentrés par l'utilisateur et qui sont simultanément appelés dans la même phase. Il faut d'abord remplir le tableau $patterns$ dans le fichier **ProjetSpark.java**. A partir de ce tableau on va filtrer le **JavaRDD** contenant l'ensemble des données pour ne garder que les phases qui contiennent ces 4 motifs d'accès simultanément. Ensuite on va utiliser la fonction $map()$ pour que chaque donnée de notre nouveau **JavaRDD** contienne le début de la phase (exprimée sous forme de Date) et la fin de la phase (exprimée sous la forme de Date) séparées par un caractère "-".

Grâce à ces données, on effectue ensuite un $flatmap()$ pour créer plusieurs plages horaires lorsque le date de début et de fin de phase est sur plusieurs plages horaires. Pour éviter de se retrouver avec plusieurs fois les mêmes valeurs on utilise la fonction $distinct()$.

2.8 Question 8

3 Résultats obtenus

N'étant plus possible de faire tourner nos programmes sur les données présentes dans le dossier **/raw_data/ALCF_repo/** pour une majorité de groupes sur le Cluster à partir du 15/01. Il a été choisi d'effectuer les opérations de traitement de données sur un plus petit fichier. Il s'agit du fichier **phases.csv** présent à la racine du projet. Les réponses aux questions concernent donc ce fichier. Dans ce fichier, il n'y a que 11 motifs d'accès différents (0, 1, 2, 3, 4, 5, 6, 7, 8, 9 et 10) et 7 jobs différents (1, 2, 3, 4, 5, 8 et 11). N'ayant que peu de records dans notre fichier, Notre classe **TopK** effectue un Top 3 et non un Top 10.

3.1 Question 1

3.1.a .....

NON IDLE PHASES DURATION

```
count is = 31
min is = 1059
max is = 4008314924
average is = 825319474
median is = 264679
1st quartile is = 2926
3rd quartile is = 33934861
histogram is = [
  1059-801663832:24
  801663832-1603326605:0
  1603326605-2404989378:0
  2404989378-3206652151:1
  3206652151-4008314924:6
]
```

3.1.b .....

IDLE PHASES DURATION

```
count is = 5
min is = 3306
max is = 23973604
average is = 4883152
median is = 3453
1st quartile is = 3352
3rd quartile is = 432045
histogram is = [
  3306-4797365:4
  4797365-9591425:0
  9591425-14385484:0
  14385484-19179544:0
  19179544-23973604:1
]
```

3.1.c .....

PATTERN 0 DURATION

```
count is = 2
min is = 3278
max is = 234327
average is = 118802
median is = 118802
1st quartile is = 3278
3rd quartile is = 234327
histogram is = [
  3278-49487:1
  49487-95697:0
  95697-141907:0
  141907-188117:0
  188117-234327:1
]
```

PATTERN 1 DURATION

```
count is = 1
min is = 1231
max is = 1231
average is = 1231
median is = 1231
1st quartile is = 1231
3rd quartile is = 1231
histogram is = [
  1231-1231:0
  1231-1231:0
  1231-1231:1
  1231-1231:0
  1231-1231:0
]
```

PATTERN 2 DURATION

```
count is = 4
min is = 1147
max is = 3828074631
average is = 957120012
median is = 202136
1st quartile is = 4850
3rd quartile is = 3828074631
histogram is = [
  1147-765615843:3
  765615843-1531230540:0
  1531230540-2296845237:0
  2296845237-3062459934:0
  3062459934-3828074631:1
]
```

PATTERN 3 DURATION

```
count is = 1
min is = 33934861
max is = 33934861
average is = 33934861
median is = 33934861
1st quartile is = 33934861
3rd quartile is = 33934861
histogram is = [
  33934861-33934861:0
  33934861-33934861:0
  33934861-33934861:1
  33934861-33934861:0
  33934861-33934861:0
]
```

PATTERN 4 DURATION

```
count is = 5
min is = 1132
max is = 24818190
average is = 5016243
median is = 3359
1st quartile is = 2926
3rd quartile is = 255610
histogram is = [
  1132-4964543:4
  4964543-9927955:0
  9927955-14891366:0
  14891366-19854778:0
  19854778-24818190:1
]
```

PATTERN 5 DURATION

```
count is = 3
min is = 3432724546
max is = 4008314924
average is = 3756816050
median is = 3829408680
1st quartile is = 3432724546
3rd quartile is = 4008314924
histogram is = [
  3432724546-3547842621:1
  3547842621-3662960697:0
  3662960697-3778078772:0
  3778078772-3893196848:1
  3893196848-4008314924:1
]
```

PATTERN 6 DURATION

```
count is = 7
min is = 1098
max is = 3451309932
average is = 912567798
median is = 3193
1st quartile is = 1212
3rd quartile is = 2911310877
histogram is = [
  1098-690262864:5
  690262864-1380524631:0
  1380524631-2070786398:0
  2070786398-2761048165:0
  2761048165-3451309932:2
]
```

PATTERN 7 DURATION

```
count is = 3
min is = 288028
max is = 25988406
average is = 12734656
median is = 11927534
1st quartile is = 288028
3rd quartile is = 25988406
histogram is = [
  288028-5428103:1
  5428103-10568179:0
  10568179-15708254:1
  15708254-20848330:0
  20848330-25988406:1
]
```

PATTERN 8 DURATION

```
count is = 1
min is = 264679
max is = 264679
average is = 264679
median is = 264679
1st quartile is = 264679
3rd quartile is = 264679
histogram is = [
  264679-264679:0
  264679-264679:0
  264679-264679:1
  264679-264679:0
  264679-264679:0
]
```

PATTERN 9 DURATION

```
count is = 1
min is = 5130
max is = 5130
average is = 5130
median is = 5130
1st quartile is = 5130
3rd quartile is = 5130
histogram is = [
  5130-5130:0
  5130-5130:0
  5130-5130:1
  5130-5130:0
  5130-5130:0
]
```

PATTERN 10 DURATION

```
count is = 1
min is = 271172
max is = 271172
average is = 271172
median is = 271172
1st quartile is = 271172
3rd quartile is = 271172
histogram is = [
  271172-271172:0
  271172-271172:0
  271172-271172:1
  271172-271172:0
  271172-271172:0
]
```

3.2 Question 2

NON IDLE PHASES NPATTERNS

```
count is = 31
min is = 1
max is = 4
average is = 1
median is = 1
1st quartile is = 1
3rd quartile is = 1
histogram is = [
  1-1:29
  1-2:0
  2-2:0
  2-3:0
  3-4:2
]
```

3.3 Question 3

NON IDLE PHASES NJOBS

```
count is = 31
min is = 1
max is = 5
average is = 1
median is = 1
1st quartile is = 1
3rd quartile is = 2
histogram is = [
  1-1:20
  1-2:7
  2-3:3
  3-4:0
  4-5:1
]
```

3.4 Question 4

3.4.a .....

TOTAL TIME JOBS DURATION

```
count is = 7
min is = 3829408680
max is = 14620138499
average is = 8431284993
median is = 7294976493
1st quartile is = 4008314924
3rd quartile is = 11358686721
histogram is = [
  3829408680-5987554643:2
  5987554643-8145700607:2
  8145700607-10303846571:0
  10303846571-12461992535:2
  12461992535-14620138499:1
]
```

3.4.b .....

TOP K TOTAL TIME JOBS

```
1. 2   14620138499
2. 11   11358686721
3. 3   11168077844
```

3.5 Question 5

IDLE PHASES TOTAL TIME

```
total duration is = 0.006782155555555555 hours
```

3.6 Question 6

3.6.a .....

PATTERN 0	PATTERN 1
solo time percent is = 9.286921E-4% simultaneous time percent is = 0.0%	solo time percent is = 4.811431E-6% simultaneous time percent is = 0.0%
PATTERN 2	**PATTERN 3**
solo time percent is = 14.96382434% simultaneous time percent is = 15.6%	solo time percent is = 0.132636266% simultaneous time percent is = 0.0%
PATTERN 4	**PATTERN 5**
solo time percent is = 0.098031312% simultaneous time percent is = 0.0%	solo time percent is = 44.05116500% simultaneous time percent is = 15.6%
PATTERN 6	**PATTERN 7**
solo time percent is = 24.96774918% simultaneous time percent is = 0.0%	solo time percent is = 0.149322305% simultaneous time percent is = 0.0%
PATTERN 8	**PATTERN 9**
solo time percent is = 0.0010345123% simultaneous time percent is = 15.6%	solo time percent is = 2.0050886E-5% simultaneous time percent is = 15.6%
PATTERN 10	
solo time percent is = 0.001059890% simultaneous time percent is = 0.0%	

3.6.b .....

TOP K TOTAL TIME PATTERNS
1. 5 11270448150
2. 6 6387974586
3. S9 4000001059

3.7 Question 7

PATTERNS PER HOURS
20h−21h 21h−22h

4 Bilan

Au cours de ce projet, nous avons appris plusieurs choses. La première est l'utilisation de Spark pour traiter des données importantes. Ensuite nous avons également appris l'impact que pouvait avoir ce type d'outil sur un Cluster. Les machines concernées subissent un important ralentissement. Lorsque les ressources sur un ensemble de machines sont partagées par tout un groupe, il est important de les partager pour que chacun puisse avancer. Nous avons donc en plus de cela pratiqué la mise en place de compromis.

La principale critique que nous pouvons émettre à propos de notre rendu et de ne pas avoir réussi à diviser notre projet en 2 parties distinctes pour chaque application que nous utilisons. Devoir à chaque fois décommenter et recommenter une partie du fichier **pom.xml** pour compiler séparément nos applications est pénible. Notre second regret est de ne pas avoir réussi à tester l'application **PhaseSpark** sur l'ensemble de données stocké dans le répertoire **/raw_data/ALCF_repo**. Il aurait été vraiment intéressant d'avoir des résultats et d'avoir un retour sur les performances de nos algorithmes à partir de ces données.