Algorithme pour le Big Data

Compte rendu de TPs

Etudiants-Rédacteurs :

Yoann Boyère - Alexis Brault

Décembre 2017

Sommaire

Objectif du TP

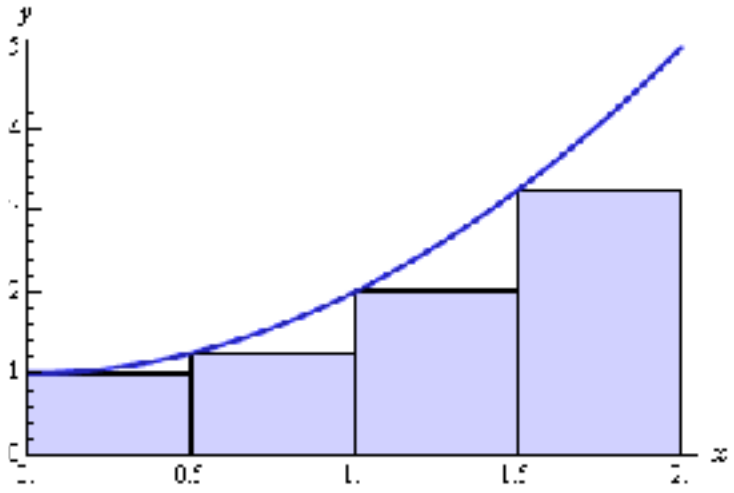
Le but de ce TP était de faire une introduction sur le framework Spark en partant d'une seule machine jusqu'au déploiement d'un cluster de 2 machines et en mettant en place un HDFS.

Part I of the lab : single machine experiment

Computing an integral

$$\int_1^{10} \frac{1}{x} dx$$

La valeur mathématiques de cette intégrale est ln(10). Pour cela nous avons calculé la somme des rectangles en prenant leur hauteur respective.



Voici notre code :

```python
from __future__ import print_function

import sys
import numpy as np
from random import random
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("BRAULT_BOYERE_INTEGRALE")\
        .getOrCreate()

    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = float(sys.argv[2]) if len(sys.argv) > 2 else 81*81*81

    a = 1
    b = 10

    pas = (b-a)/n

    def f(_):
        return ((1./_)*pas)

    count = spark.sparkContext.parallelize(np.arange(1, b, pas), partitions).map(f).reduce(add)

    print("Integrate 1/x from 1 to 10 is %f" % (count))

spark.stop()
```

Pour chaque échantillon, nous avons fais une moyenne de 5 valeurs. Nous avons fais varier les 3 paramètres plusieurs fois.

Temps d'exécution avec les différents paramètres :

Number of partitions(slices)	Temps exécution(Avg)
1	0,753286 s
2	0,738604 s
5	1,816426 s

Number of rectangles	Temps exécution(Avg)
1	0,734625 s
100	0,675215 s
10000	0,835545 s
1000000	1,090608 s

Number workers	Temps exécution(Avg)
1	0,737637 s
2	0,853584 s

Nous pouvons constater que le nombres de partitions va grandement influer sur le temps d'exécution. En effet, le nombre optimal de partitions se situe entre 2 et 5. Après, le temps de cette exécution va augmenter. Il faut comprendre que la synchronization entre les différentes partitions est trop longue par rapport au temps de calcul (même explication pour le nombre de workers (slaves)). Il s'agit du même problème que celui des peintre vu en cours. Pour ce qui est des rectangles, plus nous en ajoutons et plus le temps sera long (sauf de 1 à 100).

Analyzing a (small) geolocated social dataset

Code pour la question 1 et 2 (avec système de fichier hadoop):

```
from pyspark import SparkContext, SparkConf

def parseLine(line):
        searchObj = re.search( r'(([0-9A-z]+),([0-9]+) ?)+', line)
        if searchObj:
                return searchObj.group().split(" ")
        else:
                return  ""


if __name__ == "__main__":

        conf = (SparkConf()
         .setMaster("local")
         .setAppName("BRAULT_BOYERE_COUNTWORD")
         .set("spark.executor.memory", "1g"))
        sc = SparkContext(conf = conf)

        shutil.rmtree('/spark-2.2.0-bin-hadoop2.7/wordsOut')

        #text_file = sc.textFile("file:///spark-2.2.0-bin-hadoop2.7/words.txt")
        text_file = sc.textFile("hdfs://esir-abd-boyere-brault.istic.univ-rennes1.fr:8020/user/hadoop/words.txt")


        counts = text_file.flatMap(parseLine) \
             .map(lambda word: (word.split(",")[0], int(word.split(",")[1])))\
             .reduceByKey(lambda a, b: a + 1)\
             .sortBy(lambda a: a[1],False)
        #reduceByKey(lambda a, b: a + b) pour la question 1
        text_file.saveAsTextFile("file:///spark-2.2.0-bin-hadoop2.7/wordsOut")
sc.stop()
```

Temps d'exécution avec les différents paramètres pour les questions 1 et 2 (avec système de fichier hadoop):

Number of partitions	Temps exécution(Avg)
1	1,536748 s
2	0,972325 s
5	1,091704 s

Number of workers	Temps exécution(Avg)
1	1,017634 s
2	1,536748 s

L'utilisation d'un worker supplémentaire n'est pas nécessaire. Qui plus est, le nombre de partition utilisé varie beaucoup mais il semble qu'il serait préférable d'utiliser plusieurs partitions plutôt qu'une seule.

Code pour la question 3 (avec la moyenne par utilisateur par localisation):

```
def parseLine(line):
        searchObj = re.search( r'(([0-9A-z]+),([0-9]+) ?)+', line)
        if searchObj:
                return searchObj.group().split(" ")
        else:
                return  ""


if __name__ == "__main__":

        conf = (SparkConf()
         .setMaster("local")
         .setAppName("BRAULT_BOYERE_COUNTWORD")
         .set("spark.executor.memory", "1g"))
        sc = SparkContext(conf = conf)

        shutil.rmtree('/spark-2.2.0-bin-hadoop2.7/wordsOut')
        text_file = sc.textFile("file:///spark-2.2.0-bin-hadoop2.7/words.txt")
        map = text_file.flatMap(parseLine) \
             .map(lambda word: (word.split(",")[0], int(word.split(",")[1])))\
              .map(lambda nameTuple: (nameTuple[0], [ nameTuple[1] ])) \
                .reduceByKey(lambda a, b: a+b)\
                .mapValues(lambda x: float(sum(x))/float(len(x)))\
                .sortBy(lambda a: a[1],False)


        map.saveAsTextFile("file:///spark-2.2.0-bin-hadoop2.7/wordsOut")
sc.stop()
```

Temps d'exécution avec les différents paramètres pour la question 3 :

NbPartitions	Temps exécution(Avg)
1	2,641691 s
2	2,836259 s

5	2,840194 s

NbWorkers	Temps exécution(Avg)
1	3,926805 s
2	2,641691 s

On peut voir que d'augmenter le nombre de workers devient intéressant lorsque les calculs deviennent assez lourds comme dans la question 3. SInon, le nombre de partitions ne va pas vraiment permettre de réduire le temps d'exécution du programme.

Part II of the lab : cluster experiment

Nous avons donc ajouter une autre machine sur lequel nous avons aussi installé spark.

Ensuite, il nous suffit de lancer cette commande pour l'ajouter en tant que "slave" au cluster : `./sbin/start-slave.sh spark://ESIR-ABD-BOYERE-BRAULT.istic.univ-rennes1.fr:7077`

Pour confirmons que notre cluster de deux machines est bon, nous pouvons regarder sur l'UI :

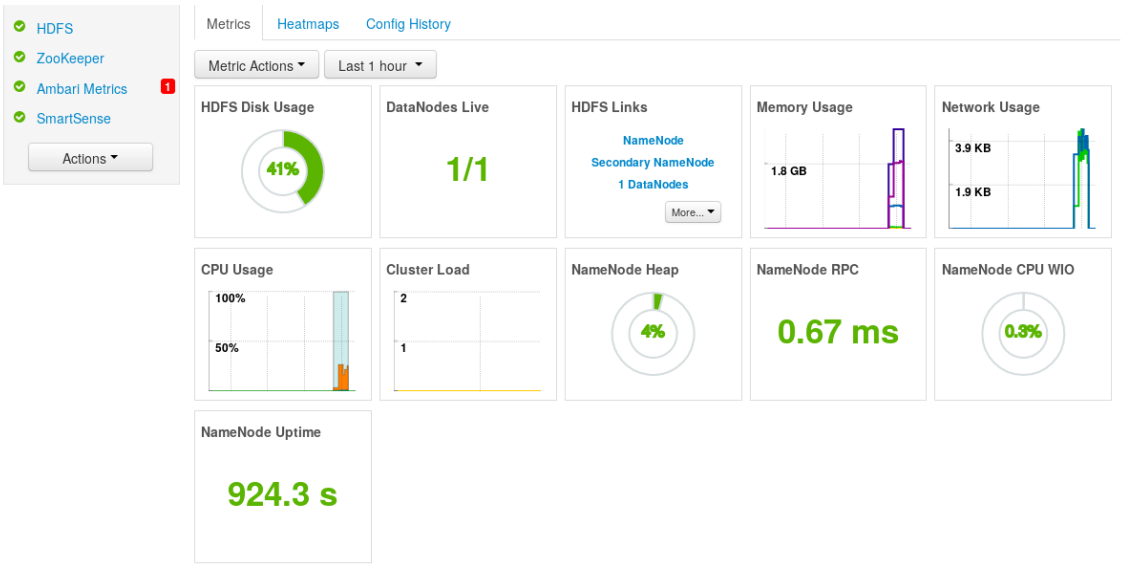
Workers

Worker Id	Address	State
worker-20171206130436-148.60.11.247-41265	148.60.11.247:41265	ALIVE
worker-20171206130622-148.60.11.226-34739	148.60.11.226:34739	ALIVE

Nous pouvons constater que nous avons bien nos deux workers. Les temps d'exécution avec un cluster de deux machines sont déjà présentés dans les tableaux des workers dans la partie précédente.

Part III of the lab : cluster experiment with hdfs

Pour faire la mise en place d'hadoop, nous avons utilisé le framework ambari. Voici notre dashboard :



Vous pouvez voir dans le code de la question 1 et 2 de la partie "Analyzing a (small) geolocated social dataset" que nous avons donc récupéré le fichier words.txt en tant que fichier Hadoop. Nous avions au préalable intégré ce fichier dans le système de fichier hadoop via la commande *hadoop fs -put*.

Conclusion

Nous avons donc utilisé Spark pour faire des calculs ainsi qu'une utilisation de grandes masses de données. Il est performant. En ajoutant une seconde machine, nous avons remarqué que le temps d'exécution était beaucoup plus efficace lorsque que les calculs commençaient à être très important. L'utilisation d'HDFS permet aussi la réduction du temps d'accès et de traitement des fichiers.