

Paralel KMeans, Hadoop

K-Means algoritmasını nasıl paralel şekilde işletiriz? Özellikle Hadoop gibi bir Esle-İndirge (Map-Reduce) ortamını düşünelim. Veri çok büyük ölçekte olabilir ve bu veriler birden fazla makineye bölünecektir. Esle-İndirge kavramında esleme safhasında “anahtar üretiriz”, ve sonra indirgeme safhasında Hadoop sistemi öyle kurmuştur ki aynı anahtarlar tek bir makineye gönderilir, ve bu nihai aşamada artık anahtar bazında indirgeme (özetleme) yapılır.

Paralel K-Means için anahtar nedir?

Anahtar, mesela küme olabilir. Yani küme 1, küme 2 gibi küme işaretleri / sayıları anahtar olarak kullanılabilirler.

Peki anahtar ile eslenecek “değer” nedir?

Öyle bir değer arıyoruz ki üst üste konulabilecek bir şey olmalı, çünkü EI sisteminin kuvveti burada, anahtarlar farklı noktalarda üretilebiliyor, sonra tek noktada üst üste konuyor, o zaman değerler öyle üretilmeli ki bu üst üste koyma, özetleme işlemi yapılabilir.

Üst üste konabilecek şey kümeye (anahtar) ait olan veri noktası olabilir. Normal K-Means’i hatırlarsak, her nokta için o noktaya en yakın kümeyi buluyordu sonra, atama işlemi bitince, her kümenin altındaki noktaların toparlayıp, onların ortalamasını alarak yeni küme merkezini hesaplıyordu. Bu ortalama işlemi üst üste konabilecek bir şey, yani farklı makinelerde küme-nokta eşlemelerini üretirsek, indirgeme aşamasında o anahtar için tüm değerleri toplayıp, nokta sayısına böleriz ve yeni küme merkezini elde ederiz.

```
from IPython.display import Image
Image(filename='kmeans-diag.png')
```

```
<IPython.core.display.Image at 0xa84e4ac>
```

Şimdi Hadoop ile ilgili bazı lojistik konulara gelelim:

Eğer esleme safhasında her nokta için en yakın kümeyi bulmak istiyorsak, o zaman (ilk başta rasgele bile olsa) küme merkezlerinin bilgisi tüm makinelerin erişebileceği bir yerde olmalı. Biz bu veriyi HDFS üzerinde tutmaya karar verdik, dosya ismi bilinen (well-known) bir yerde olacak, /tmp/centers.csv.

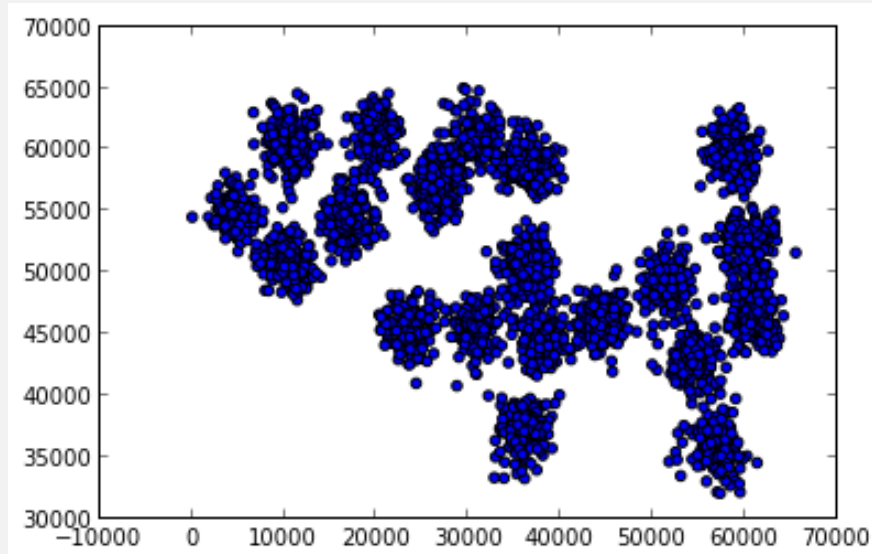
Paralel K-Means için tek bir esle-indirge işletimi yeterli değil, bu algoritma döngülü / özyineli (iterative) bir algoritma, 5,10,20 kez işlemesi gerekebilir. Her döngü (indirgeme) sonunda yeni küme merkezleri hesaplanacak, bu merkezler eski /tmp/centers.csv yerini alacak ve işlem tekrar başlayacak.

Döngü sonundaki merkez bilgisi indirgeyicinin çıktısıdır ve bu çıktı output/part-00000 dosyası içinde. Unutmayalım, indirgeyicinin çıktısı demek, tüm indirgeyici makinelerden gelen anahtarların (özetleme ardından) birleştirilmesi sonrasında demek.

Şimdi ham veriyi gösterelim, ve Hadoop’u başlatalım.

```
from pandas import *
df1 = read_csv("synthetic.txt", sep=" ")
plt.scatter(df1.ix[:,0], df1.ix[:,1])
```

<matplotlib.collections.PathCollection at 0xafa270c>



```
!ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/stop-all.sh
!ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/start-all.sh
```

no jobtracker to stop

localhost: no tasktracker to stop

no namenode to stop

localhost: no datanode to stop

localhost: no secondarynamenode to stop

starting namenode, logging to /home/hduser/Downloads/hadoop-1.0.4/libexec/../../logs/hadoop-hduser-namenode

localhost: starting datanode, logging to /home/hduser/Downloads/hadoop-1.0.4/libexec/../../logs/hadoop-hduser

localhost: starting secondarynamenode, logging to /home/hduser/Downloads/hadoop-1.0.4/libexec/../../logs/hadoop-hduser

starting jobtracker, logging to /home/hduser/Downloads/hadoop-1.0.4/libexec/../../logs/hadoop-hduser-jobtra

localhost: starting tasktracker, logging to /home/hduser/Downloads/hadoop-1.0.4/libexec/../../logs/hadoop-h

Gerekli merkez verisinin tutulacagi yer /tmp demistik. Bu ismi ozellikle sectik cunku hem yerel, komut satirindan (Hadoop olmadan) calisirken rahat kullanilabilecek bir dizin olsun istedik. Bu dizini HDFS uzerinde yaratalim (unutmayalim, HDFS demek ayri bir alem demek, pur Unix komutlarimiz bile oraya erisemiyor)

```
!ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/hadoop dfs -mkdir /tmp/
```

```
mkdir: cannot create directory /tmp/: File exists
```

```
!ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/hadoop dfs -ls /
```

```
Found 3 items
```

```
drwxr-xr-x - hduser supergroup      0 2013-02-25 17:23 /app
drwxr-xr-x - hduser supergroup      0 2013-02-26 12:49 /tmp
drwxr-xr-x - hduser supergroup      0 2013-02-26 11:45 /user
```

```
!ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/hadoop dfs -copyFromLocal
    $HOME/Documents/classnotes/stat/stat_hadoop_kmeans/synthetic.txt /user/hduser
```

```
copyFromLocal: Target /user/hduser/synthetic.txt already exists
```

```
print open("mapper.py").read()
```

```
#!/usr/bin/python
import os,sys,itertools
import numpy as np
from numpy import linalg as la
os.environ['MPLCONFIGDIR']='/tmp'
import pandas as pd

centers = pd.read_csv("/tmp/centers.csv",header=None,sep=",")

def dist(vect,x):
    return np.fromiter(itertools.imap(np.linalg.norm, vect-x),dtype=np.float)

def closest(x):
    d = dist(np.array(centers)[: ,1:3],np.array(x))
    return np.argmin(d)

comb = lambda x: str(x[0])+":"+str(x[1])

df = pd.read_csv(sys.stdin,header=None,sep=" ")
df['cluster'] = df.apply(closest,axis=1)
df['coord'] = df.apply(comb,axis=1)
df.to_csv(sys.stdout, sep='\t',index=False, cols=['cluster','coord'],
    header=None)
```

Ustte comb ile kordinat verisini birlestirerek tek kolon haline getirdik, cunku anahtar deger formunda veri uretmemiz gerekiyor, ve TAB ayraci sadece tek anahtar ve tek deger ayrimi yapabilir, ve sadece bir tane ayrac olabilir. Bu sebeple iki boyutlu veri oldugu icin iki sayi olan deger, yani kordinat : uzunden birlestirilerek tek bir deger haline getirildi. Tabii indirgeci bu veriyi alinca bu islemin tersini yapmasi lazim.

```
print open("reducer.py").read()

#!/usr/bin/python
import os,sys,itertools
import numpy as np
from numpy import linalg as la
os.environ['MPLCONFIGDIR']='/tmp'
import pandas as pd

def coords(x):
    return pd.Series(np.array(str(x).split(":"),dtype=np.float64))

df = pd.read_csv(sys.stdin,sep="\t",names=['cluster','coord'])
df2 = df['coord'].apply(coords)
df3 = df.combine_first(df2)
df4 = df3.groupby('cluster').mean()
df4.to_csv(sys.stdout, sep=',',header=None)
```

Alttaki blokta merkezleri rasgele seciyoruz. Bunun icin veri icinden rasgele 15 tane (k kadar) nokta cekip cikarttik. Bu rasgele sayi uretimi ile de yapilabilirdi.

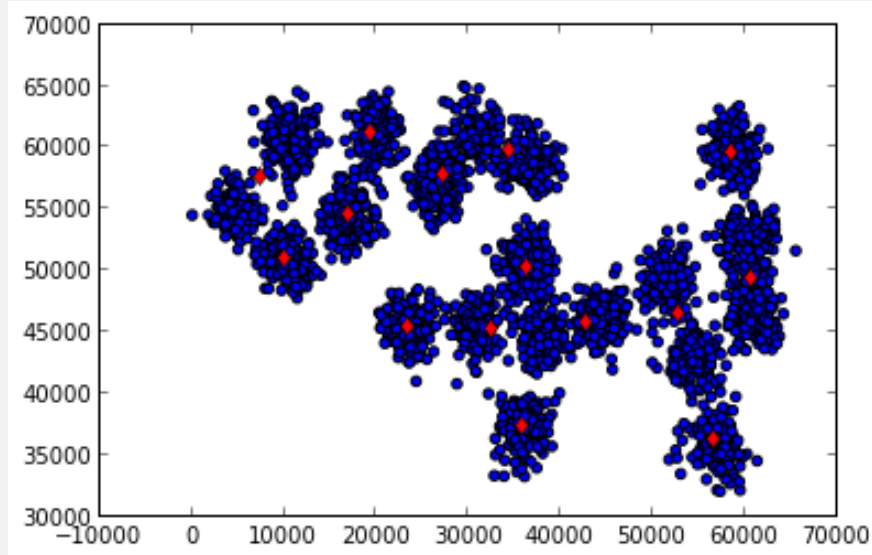
```
import os,sys,itertools
from numpy import linalg as la
import pandas as pd
k = 15
df = read_csv("synthetic.txt",header=None,sep=" ")
centers = df.take(np.random.permutation(len(df))[:k])
centers.to_csv("/tmp/centers.csv",header=None)
```

Komut satirindan tek makina icin Hadoop'suz isletelim,

```
for i in range(20):
    os.system("cat synthetic.txt | python mapper.py | python reducer.py > /tmp/centers2.csv")
    os.system("mv /tmp/centers2.csv /tmp/centers.csv")
```

```
df1 = read_csv("synthetic.txt",sep=" ")
plt.scatter(df1.ix[:,0],df1.ix[:,1])
plt.hold(True)
df2 = read_csv("/tmp/centers.csv",header=None,index_col=0)
plt.plot(df2.ix[:,1],df2.ix[:,2], 'rd')
```

[<matplotlib.lines.Line2D at 0xb1ff8cc>]



Simdi K-Means'i Hadoop'tan isleten kosturucu (runner) program. Bu program esleme, indirgeme safhalarini cagiracak, ve indirgeyici sonrasi ciktiyi alip yeni kume merkezi yapacak, bunun gibi idari / temizlik islerini halledecek.

```
os.system("cp mapper.py /tmp/")
os.system("chmod a+r /tmp/mapper.py")
os.system("chmod a+x /tmp/mapper.py")

os.system("cp reducer.py /tmp/")
os.system("chmod a+r /tmp/reducer.py")
os.system("chmod a+x /tmp/reducer.py")

hp_cmd = "ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/hadoop"
streaming_cmd = "/home/hduser/Downloads/hadoop*/contrib/streaming/hadoop-*streaming*.jar"

# Ilk seferde merkez verisini HDFS'e yerel dizinden gonder
os.system("%s dfs -rm /user/hduser/centers.csv" % hp_cmd)
os.system("%s dfs -copyFromLocal /tmp/centers.csv /user/hduser" % hp_cmd)

for i in range(20):
    os.system("%s dfs -rmr /user/hduser/output" % hp_cmd)
    os.system("%s jar %s -input synthetic.txt -output output -mapper /tmp/mapper.py -"
              "reducer /tmp/reducer.py -numReduceTasks 1" % (hp_cmd,streaming_cmd))
    os.system("%s dfs -cp /user/hduser/output/part-00000 /user/hduser/centers.csv" %
              hp_cmd)
```

Ustte K-Means'i 20 kere islettigimizi goruyoruz. Eger istenirse (hatta daha iyi olur) dongu bir while

icine konur ve bitis icin “stabilite sarti” aranir. Stabilite yeni kume merkezinin eskisinden “cok fazla degisik olup olmadigi” sartidir, degisim yoksa artik sonucu bulmusuz demektir, daha fazla donguye gerek kalmayacaktır. Biz donguyu 20 kere donguyu islettik, (bu problem icin) yeterli oldu.

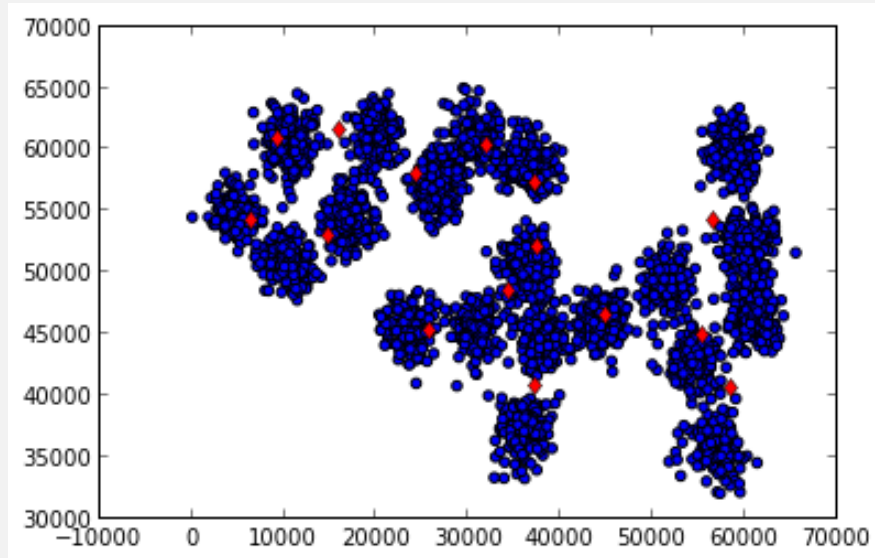
K-Means isini bitirdikten sonra elde edilen sonuclari artik HDFS’ten yerel dizinimize alabiliriz. Nihai kume merkezleri output/part-00000 icinde. Bu merkezleri alip, ham veri uzerinde kirmizi nokta olarak gosteriyoruz.

```
os.system("ssh localhost -l hduser rm /tmp/part-00000")
os.system("ssh localhost -l hduser /home/hduser/Downloads/hadoop*/bin/hadoop dfs -
copyToLocal /user/hduser/output/part-00000 /tmp/") #
```

0

```
df1 = read_csv("synthetic.txt",sep=" ")
plt.scatter(df1.ix[:,0],df1.ix[:,1])
plt.hold(True)
df2 = read_csv("/tmp/part-00000",header=None,index_col=0)
plt.plot(df2.ix[:,1],df2.ix[:,2], 'rd')
```

[<matplotlib.lines.Line2D at 0xb369d6c>]



Sonuclar fena degil. Iste bu metotla terabayt olceginde, devasa bir veriyi 20-30 makinaya dagitarak parca parca isleyip kumelemeniz mumkundur. Endustride son zamanlarda habire duyulan Buyuk Veri (Big Data) olayi iste bu.