bigmemoRy

Big Data in R Project

Analyzing the IMDB data with BigMemoRy and SparkR : Comparative Approach

Submitted by
Moustapha OUSMANE BAWA GAOH
Shishir DUBEY
Shashank SHARMA

Table of Contents

INTRODUCTION

I. Challenges at Large Scale

Performing large-scale computation is very difficult. To work with this volume of data requires distributing parts of the problem to multiple machines to handle in parallel. Whenever multiple machines are used in cooperation with one another, the probability of failures rises. In a single-machine environment, failure is not something that program designers explicitly worry about very often: if the machine has crashed, then there is no way for the program to recover anyway. In a distributed environment, however, partial failures are an expected and common occurrence. Synchronization between multiple machines remains the biggest challenge in distributed system design. When it comes to Big Data this proportion is turned upside down. Big Data comes into play when the CPU time for the calculation takes longer than the cognitive process of designing a model. The number of records of a data set is just a rough estimator of the data size though. It's not about the size of the original data set, but about the size of the biggest object created during the analysis process. Depending on the analysis type, a relatively small data set can lead to very large objects.

II. Main Perspective and Goal to achieve

Setting up the Hadoop Cluster which enables to Synchronize the work and allow the parallelization between the multiple nodes and which must lead to low failure of resources and allows integration of Spark and R for making some Analysis on the data available and fetching out the predictions with the proper utilization of each and every node connected within the cluster.

Using bigmemoRy Approach to resolve the constraints which pops up while dealing with large matrices on R that can be massive data sets (perhaps requiring several GB of memory on typical computers) but not larger than the total available RAM which will be utilized to deal with this kind of situation and allows to deal with huge data as much faster as possible .

Data Extraction and Tools used

The data used for the project is been extracted from the available open Data Source of IMDB which was compressed in the form of plain text file with the extension (.list) and was extracted using different tools that are:-

I. Python

It was used to extract the different fields of the data and this data is been saved in the form of sqlite3 database file .The database contains these specific Tables .The specific tables were:

(i)Actors
(ii)Acted_in
(iii)movies
(iv)movies_genres
(v)movies_keywords
(vi)Series
(vii)Genres

II. SQL

It was used to merge the distinct fields with the help of structured query language and Applying Joins to get the structured schema and the big table was formed and converted to csv with the following fields were taken into account :

(i)ActorsId
(ii)l_name
(ii)m_name
(iii)f_name
(iv)idmovies
(v)title
(vi)years
(vii)location
(viii)id genre
(ix)genre
(x)id series
(xi)series name
(xii)season

Requirements and their importance in implementing Hadoop

Some Softwares and Configurations required before setting up Hadoop cluster .These are as follows :-

I. Java

Proper version of java must be installed in order to be used with respective version of Hadoop. We used jdk1.7 for the hadoop implementation.

Why Java?
Because Hadoop itself is written in Java and all its components such as map-reduce, zoo keeper ,Hbase,HDFS requires Java because at some point they uses Java API's and so its necessary installing Java before setting up hadoop cluster .

II. SSH Client

ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.

III. Hadoop

The suitable version of hadoop must be installed which supports the Appropriate Java version available .We used Hadoop Version 2.7.2

IV. Configurations

- The path to the Java must be set among Environment variables in order to not to set the path again and again if every time java is required whenever the Hadoop is used or cluster is implemented.

- The ssh keys must be created in order to establish connection between the Nodes with the help of passing public key to the remote server and establishing the Mutual understanding between them .The Path to hadoop must also be set among environment variables in order to utilize hadoop components from anywhere on the local host in order to access , copy, fetch and make many other operations between local host and hdfs.

Implementation of Hadoop cluster on Single Node Instance

As Hadoop on the single node cluster uses two approaches to be settled up:-

I. Standalone Mode

By default, Hadoop is configured to run in a non-distributed or standalone mode, as a single Java process. There are no daemons running and everything runs in a single JVM instance. HDFS is not used .We don't have to do anything as far as configuration is concerned, except the JAVA_HOME. Therfore, single node setup is one where you have (presumably) one datanode and one tasktracker on a single machine.

II. Pseudo Distributed Model

The Hadoop daemons run on a local machine, thus simulating a cluster on a small scale. Different Hadoop daemons run in different JVM instances, but on a single machine. HDFS is used instead of local FS.As far as pseudo-distributed setup is concerned, you need to set at least following 2 properties along with JAVA_HOME:

1.fs.default.name in core-site.xml
2.mapred.job.tracker in mapred-site.xml

we could have multiple datanodes and tasktrackers, but that doesn't make much sense on a single machine.
Therefore,Single Node or Psuedo-Distributed Cluster is the one in which all the essential daemons (like NameNode, DataNode, JobTracker and TaskTracker) run on the same machine. The default replication factor for a single node cluster is 1. A single node cluster is basically used to simulate a full cluster like environment and to test hadoop applications and unlike the Stand-alone mode, HDFS is accessible in the Pseudo distributed mode.
This was implemented in order to test if every Daemon is running on our Machine.

Implementation of Hadoop cluster on Multi-Node Instances and Tools Implemented

I. Fully Distributed Mode

Multi Node or Fully Distributed Cluster is a typical hadoop cluster which follows a master-slave architecture. It will basically comprise of one master machine (running the NameNode and TaskTracker daemon) and one or more slave machines (running the DataNode and TaskTracker daemon). The default replication factor for a multi node cluster is 3. It is basically used for full stack development of hadoop application and projects.

i. Configurations Done:

Hadoop configuration is driven by two types of important configuration files:

1. Read-only default configuration - src/core/core-default.xml, src/hdfs/hdfs-default.xml and src/mapred/mapred-default.xml.

2. Site-specific configuration - conf/core-site.xml, conf/hdfs-site.xml and conf/mapred-site.xml,Hadoop-env.sh

This Mode is used in order to enable the parallelization. So the different Map-Reduce Tasks were performed in order to check if the cluster is stable and distribution with parallelization between the nodes are working in a good manner.

ii. Tools Implemented :

These tools are implemented for making Data Analysis on the available IMDB database. They are as follows

1. R Programming
2. Spark
3. SparkR
4. BigmemoRy

Libraries required before using R and their importance

The most important libraries used are as follows :

I. bitops	II.RJSONIO	III.digest	IV.functional
V. Rcpp	VI.stringr	VII.plyr	VIII.reshape2
IX. codetools	X.dplyr	XI.R.methodsS3	XII.caTools
XIII. Hmisc	XIV.rmr2	XV.rJAVA	XVI.rhdfs

RHadoop packages are dependent on above packages, which should be installed for all users, instead of in personal library. Otherwise, you may see RHadoop jobs fail with an error saying "package *** is not installed"

Synchronization of R with the Hadoop Environment

Both Hadoop and R being open source, the combination of R and Hadoop seems a natural one. But, some fundamental differences between the two need to be addressed in order to make the combination work. R, on one hand, supports an iterative process beginning with a hypothesis, exploring the data, trying different statistical models, drilling down to find the exact solution. On the other hand, Hadoop supports batch processing, leaving jobs queued and executed in sequence. R is designed for in-memory, data execution while Hadoop work on a distributed setup of parallel data slices. With R and Hadoop, a robust data analytics engine can be built, which can apply algorithms to large scale dataset in a scalable manner.

So , The environment Variables HADOOP_CMD and HADOOP_STREAMING are settled up to use the hadoop within R.

Rhdfs and Rmr2 package plays an important role in synchronizing the Hadoop and R
And allows HadoopR Packages to perform their map and reduce tasks conveniently.

This implementation built up the base for the future tools used such as Spark and enabled to do some statistics within the cluster.

Implementing Big-Memory

Following steps are taken into measure:

- loading the table a first time taking arround 30 min.
- Then loading the matrix is instantaneous.
```
user  system elapsed
28.452   8.077  36.598
```
- our data was not completly loaded but we have 571 257 330 and 14
- The objective is to answer to several question that we can have through this data set.
- we will first do it on one personnal computer with bigmemory then
- compare the processing time by doing it with spark in a parllelization context

The non exhaustive list of some questions that we wanted to answer

- 1. the year of the oldest movies in the Database and the Newest ones

  Results Produced by BigMemoRy function
```
 user  system elapsed
 0.512   0.461   0.97
```
  The time taken is 10 times lesser than the normal min or max

- 2. The number of movies played by actors and the actor that played the highest number of movies

  Results Produced by R based Function
  ```
  user     system  elapsed
  103.673  32.284 136.065
  ```

  Results Produced by BigMemoRy function
  ```
  User  system elapsed
  10.685   0.418  11.208
  ```

  Like in the precedent question it appeared that the implemented bigmemoRy function was more faster than the normal R based Function .

- 3. Genre that has mostly played Roles

  Results Produced by BigMemoRy function
  ```
  user  system elapsed
  10.915   0.626  11.604
  ```

- 4. how many unique actor, genres and series we have in the database

  Results Produced by BigMemoRy function
  unique actors

  ```
  user  system elapsed
  5.971  14.812  27.700
  ```

  unique genres

  ```
  user  system elapsed
  5.760   8.419  15.820
  ```

  unique movies and unique series

  ```
  user  system elapsed
  5.797   6.125  12.836
  ```
  The series that have most seasons

  ```
  user  system elapsed
  5.714   6.154  13.186
  ```

- 5.The number of movies by year and the year that have the highest number of movies

  Results Produced by BigMemoRy function

  ```
  user  system elapsed
  10.851   0.566  11.501
  ```

Implementing sparkR package for integrating spark with R and DataAnalysis:

These are the the agregations performed on the data with the help of SparkR.

The data dimensions checking that is quasi instantanious with Bigmemory, here take around 40 second with 19 908 819 row.

Now lets try to answer to some question like in the case of bigmemory

1. The year of the oldest movie in the base and the newest one

   ```
   user  system elapsed
   0.257   0.155  48.438
   ```

   ```
   user  system elapsed
   0.188   0.167  45.659
   ```

   ```
   # 1      2017
   ```

It appear that the newest movies in the base is 2017 one's, let see this movies

2. The number of movies played by each actors and the actor that played the highest number of movies

   ```
   user  system elapsed
   1.715   0.114  38.846
   ```
   For this we remark that the actor that played the highest number of movies is "Bonnie Gail"
   And the one played highest number of series is " Abdul Paula"

3. what is the genre that is mostly played
   ```
   user  system elapsed
   0.224   0.095  41.638 3
   ```
   It appeared that the most genres played are Drama and comedy

4. how many unique actor and series we have in the database
#unique actors3

```
        user  system elapsed0.168   0.119  39.431
Count (actors) count(genres) count(seires) count(idmovies) count(location)
1      18967          27               328083          6246            105
```

5 . the series that have most seasons
## first find the highest number of season

```
user  system elapsed
1.592   0.204  39.490

max(seasons)
1      111
```

Wow there is a series that have 111 season let see which one it is

```
      idseires     series_name      years     location             title
 1    69576        Brittany Murphy  1981 weekend title Entertainment Tonight
```

5. the number of movies by year and the year that have the highest number of movies

```
topyearMovies
years numbMovieYear
63  2009          255
> topyearSeries
years numbSeries
59  2005      15162
```

So for this query it appeared that the year where there are lot of movies released is 2009 with 255 distinct movies and the year that have top series recorded is 2005 with 15162 released with the function big K means and the paralellisation lets apply the k means

Problems Faced During implementation and Failure occurred

I. First problem faced is while setting up Hadoop cluster on Multiple nodes .Some Exceptions and bugs arrived during the installation of hadoop and faced the problem with the sparkR initialization.

II. While accessing the whole IMDB database available that is of around (59.3)GB in size with the help of BigMemoRy we faced trouble that we were only able to fetch it around 36 GB and got stuck in that and fetching failure occurred.

III. While using sparkR ,we were unable to load the dataset from LocalFile system but were able to load the the database from the HDFS which was small file because there was not enough HDD memory in the system.

Conclusion

1 .Came to know about the different Approaches of dealing with the large amount of the data .

2.We were able to compare processing time between Rfunction, SparkR and BigMemoRy.

I. Key Benefits of the project

- The time processing for bigmemoRy was far better then the other two approaches ,Specially from the normal R.
- The Capacity limit constraint was removed and were able to process 59.3 Gb of data with the help of bigMemory .

II. Drawback

i. The first drawback was Loading of the data took more time around 30 minutes .
ii. Data generated is one class Data Frame or one kind of file was there ,so in our case access to the variable that are in the form of character we would have to reload the table again to change the type from integer to char .
iii. Another Drawback is there are not so many functions implemented.

Future Work

(I) In the future we will try to implement the SparkR analysis within the whole database available and among the Multinode cluster.

(II) Implementation and analysis on the Rstudio server which directly helps us to do analysis on the database available on the hdfs.

(III) Going in deep of the analysis of the database by using Using Statistical Models like Clustering to see the movies that are more related .

REFERENCES

1. The bigmemoRy package:handling large data sets in R using RAM and shared memory John W. Emerson1, Michael J. Kane
http://www.stat.yale.edu/~mjk56/Research/Prospectus/bigmemoRy-vignette.pdf

2.Taking R to the Limit (High Performance Computing in R)
http://www.slideshare.net/bytemining/r-hpc
http://www.bytemining.com/2010/08/taking-r-to-the-limit-part-ii-large-datasets-in-r/

3.Dealing with large data sets and memory issues in R
P. Lafaye de Micheaux1, ENSAI, January, 2016

4.The Map-Reduce framework in R
P. Lafaye de Micheaux1, ENSAI, January, 2016

5.Spark & R: data frame operations with SparkR
https://www.codementor.io/spark/tutorial/spark-r-data-frame-operations-sql

6.http://bigmemory.org/

7.Scalable Strategies for Computing with Massive Data: The Bigmemory Project
Michael J. Kane & John W. Emerson
http://www.slideshare.net/joshpaulson/big-memory?utm_source=slideshow02&utm_medium=ssemail&utm_campaign=share_slideshow

8.The R Package bigmemory: Supporting Efficient Computation and Concurrent Programming with Large Data Sets
John W. Emerson, Michael J. Kane
http://www.stat.yale.edu/~mjk56/temp/bigmemory-vignette.pdf

9.SparkR (R on Spark) documentation
https://spark.apache.org/docs/latest/sparkr.html

10.SparkR: Distributed data frames with Spark and R
http://blog.revolutionanalytics.com/2015/06/sparkr-announcement.html

11.Step-by-Step Guide to Setting Up an R-Hadoop System
http://www.rdatamining.com/big-data/r-hadoop-setup-guide

12.CSV Data Source for Spark
https://github.com/databricks/spark-csv

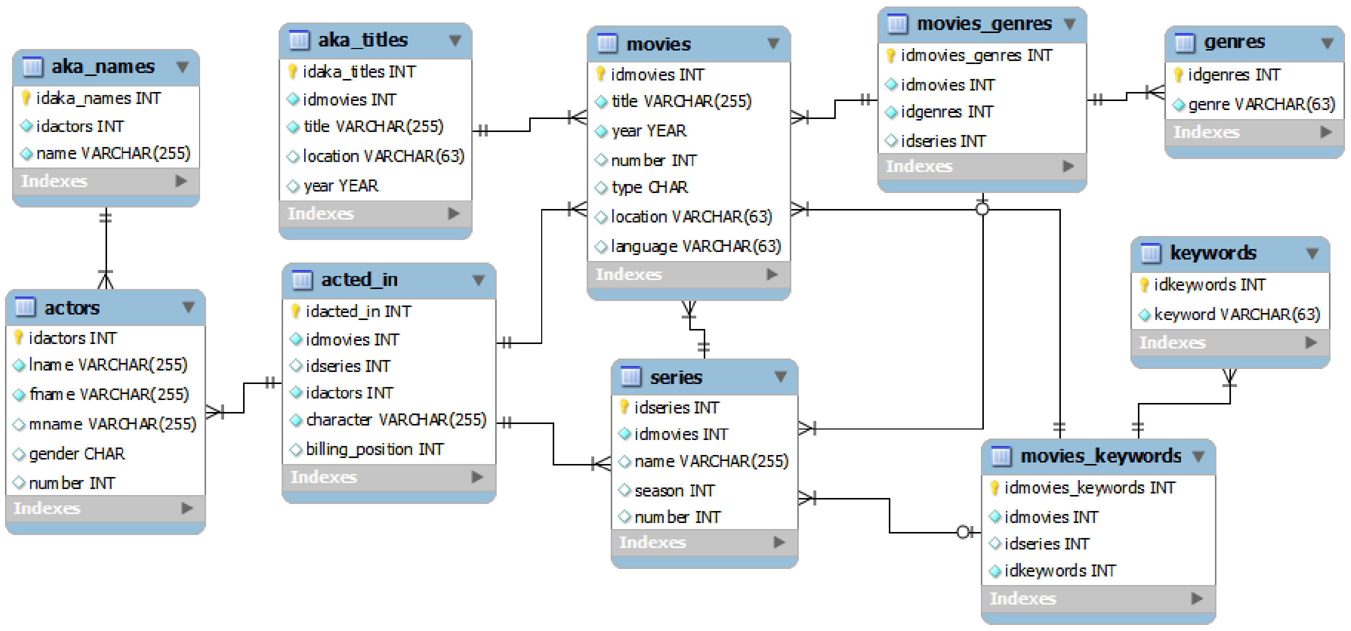
13.Spark Standalone Mode
http://spark.apache.org/docs/1.2.0/spark-standalone.html

14.Tuning Spark
http://spark.apache.org/docs/latest/tuning.html

15.The bigmemory Project
https://sites.google.com/site/bigmemoryorg/16.Where to get IMDb datasets
http://opendata.stackexchange.com/questions/1073/where-to-get-imdb-datasets

APPENDIX

I. Database Schema and Relations used



II. Rstudio server

The Rstudio Server is been settled up and integrated with hadoop in order to handle the Rscripts on the server .

For setting up Rstudio server the following measurements are taken into an account: Downloaded Rstudio Server--->Cmake---->Configurations and some manipulations in Rstudio server conf files

III. Code for BigMemoRy implementation

```
devtools::install_github("kaneplusplus/bigmemory")
devtools::install_github("kaneplusplus/bigtabulate")
devtools::install_github("kaneplusplus/biganalytics")
devtools::install_github("kaneplusplus/bigalgebra")
devtools::install_github("kaneplusplus/synchronicity")

### define the working working directory
```

```
setwd("/media/moustapha/TOSHIBA EXT/Msc ENSAI/Big Data Science Courses/Big Data with R")
###

library(bigmemory)

system.time({
X<- read.big.matrix(file.choose(),
header = TRUE, type = "integer",
backingfile = "test.bin",
descriptorfile = "test.desc")
})


# loading the table a first time taking arround 30 min
library(bigmemory)
library(biganalytics)
library(bigtabulate)
imdb <- dget("test.desc")
imdb

system.time(x <- attach.big.matrix(imdb))
## then loading the matrix is instantaniously
colnames(x)
head(x)
## head(x)
# dim(x)
# [1] 571257330        14
system.time(summary(x))
# user  system elapsed
# 28.452   8.077  36.598

## our data was not completly loaded but we have 571 257 330 and 14
head(x,240)
gg<-as.data.frame(tail(x,5000))
######
## The objective is to answer to several question that we can have through this data set.
## we will first do it on one personnal computer with bigmemory then
## compare the processing time by doing it with spark in a parllelization context

## the non exhaustive list of some question that we want the answer
## 1. the year of the oldest movie in the base and the newest one
system.time( minyear<-colmin(x,"years", na.rm = T))
system.time( maxyear<-colmax(x,"years", na.rm = TRUE))

# user  system elapsed
# 0.512   0.461   0.973

#the time take is 10 time less than the normal min or max

## 2. the number of movies played by actors and the actor that played the highest number of movies
system.time( movPlayAc<-table(x[,"idactors"]))
```

```
# user  system elapsed
# 103.673  32.284 136.065

system.time( movPlayAc<-bigtable(x,"idactors"))
# user  system elapsed
# 10.685   0.418  11.208

# like in the precedent question it appear that the implemented bigmemory function are more faster than the
normal R ones
names(which.max(movPlayAc))



## 3. what is the genre that is mostly played

system.time( movGenres<-bigtable(x,"idgenres"))
names(which.max(movGenres))

## 4. how many unique actor, genres and series we have in the database
#unique actors
system.time(uniqActor <- na.omit(unique(x[, "idactors"])))
length(uniqActor)
# unique genres
system.time(uniqGenres <- na.omit(unique(x[, "idgenres"])))
length(uniqGenres)
# unique series
system.time(uniqSeries <- na.omit(unique(x[, "idseires"])))
length(uniqSeries)
# unique movies
# unique series
system.time(uniqMovies <- na.omit(unique(x[, "idmovies"])))
length(uniqMovies)

## the series that have most seasons
system.time(maxSeason<-max(x[,"seasons"], na.rm = TRUE))
system.time(maxSeason<-colmax(x, "seasons", na.rm = TRUE))

system.time(topSeries <- x[mwhich(x, "seasons", maxSeason, "eq"),])
na.omit(unique(topSeries[,"idmovies"]))

## 5. the number of movies by year and the year that have the highest number of movies
system.time(numberMovieYear<-bigtable(x, "years"))
names(which.max(numberMovieYear))


## with the function big K means and the paralellisation lets apply the k means
require(doMC)
registerDoMC(cores = 5)
system.time(kmeanss <- bigkmeans(x,3,dist = "euclid", nstart = 5, iter.max = 100))
```

kmeanss

II. Code for SparkR:-

```
library(rJava)
library(SparkR)
Sys.setenv(SPARK_MEM="8g")
sc <- sparkR.init(sparkPackages="com.databricks:spark-csv_2.11:1.2.0")
sqlContext <- sparkRSQL.init(sc)


## path <- file.path('', 'home','moustapha','airports.csv')

system.time(imdb <- read.df(sqlContext,
"/Imdb.csv",
header='true',
source = "com.databricks.spark.csv",
inferSchema='true'))

head(imdb)
system.time(dimData=dim(imdb))

# 19908819      14

## the data dimensions checking that is quasi instantanious with Bigmemory, here take around 40
second with 19 908 819 row

## Now lets try to answer to some question like in the case of bigmemory

## 1. the year of the oldest movie in the base and the newest one
system.time( minyear<-collect(select(imdb, min(imdb$years))))
# user  system elapsed
# 0.257   0.155  48.438
# > minyear
# min(years)
# 1      1947
system.time( maxyear<-collect(select(imdb, max(imdb$years))))
# user  system elapsed
# 0.188   0.167  45.659
# > maxyear
# max(years)
# 1      2017

## it appear that the newest movies in the base is 2017 one's, let see this movies
system.time(futureMovies <-
collect(imdb[imdb$years == maxyear[[1]], c("idmovies","title","years","location","genre")]))

## that takes around 40s and it is only one movies Mockingbirds that is a mix of three genres
```

```
(adventures, mystery, romance)
# user  system elapsed
# 0.181   0.097  37.865


## 2. the number of movies played by each actors and the actor that played the highest number of
movies
system.time( movPlayAc<-collect(agg(groupBy(imdb, "idactors"),
numbMovies=countDistinct(imdb$idmovies),
numbSeries=countDistinct(imdb$idseires)
)))
## this query take 37 second

maxMov=movPlayAc[which.max(movPlayAc$numbMovies),"idactors"]
maxSeries=movPlayAc[which.max(movPlayAc$numbSeries),"idactors"]

system.time(maxMoviesActor <-
collect(imdb[imdb$idactors == maxMov, c("idactors","lname","fname","mname")]))


system.time(maxSeriesActor <-
collect(imdb[imdb$idactors == maxSeries, c("idactors","lname","fname","mname")]))

# user  system elapsed
# 1.715   0.114  38.846
##  for this  we remarque that the actor that played the highest number of movies is "Bonnie Gail"
## And the one played highest number of series is " Abdul Paula"

## 3. what is the genre that is mostly played

system.time( mostGenre<-collect(agg(groupBy(imdb, "genre"),
numbMovies=countDistinct(imdb$idmovies),
numbSeries=countDistinct(imdb$idseires))))
# user  system elapsed
# 0.224   0.095  41.638
# it appeard that the most genres played are Drama and comedy



## 4. how many unique actor and series we have in the database
#unique actors

system.time(uniqElement<-collect(select(imdb, countDistinct(imdb$idactors),
countDistinct(imdb$idgenres),
countDistinct(imdb$idseires), countDistinct(imdb$idmovies),
countDistinct(imdb$location))))
# user  system elapsed
# 0.168   0.119  39.431

# count(actors) count(genres) count(seires) count(idmovies) count(location)
# 1       18967          27       328083           6246            105
```

```
## 5 . the series that have most seasons
## first find the highest number of season
system.time(maxSeason<-collect(select(imdb, max(imdb$seasons))))
# user  system elapsed
# 1.592   0.204  39.490

# max(seasons)
# 1       111
## wow there are series that have 111 season let see which on it is

system.time(topSeries <-
collect(imdb[imdb$seasons == maxSeason[[1]], c("idseires","series_name","years","location","title")]))

# idseires     series_name years      location                title
# 1      69576 Brittany Murphy  1981 weekend title Entertainment Tonight


## 5. the number of movies by year and the year that have the highest number of movies
system.time(topYear<-collect(agg(groupBy(imdb, "years"),
numbMovieYear=countDistinct(imdb$idmovies),
numbSeries=countDistinct(imdb$idseires))))
topyearMovies<-topYear[which.max(topYear$numbMovieYear),1:2]
topyearSeries<-topYear[which.max(topYear$numbSeries),-2]
topyearMovies
topyearSeries

# topyearMovies
# years numbMovieYear
# 63  2009          255
# > topyearSeries
# years numbSeries
# 59  2005      15162

## So for this query it appeared that the year where there are lot of movies released is 2009 with 255
distinct movies
## and the year that have top series recorded is 2005 with 15162 released

## with the function big K means and the paralellisation lets apply the k means
```