Big Data Systems and Architecture

Anastasios Moraitis - p2822124

Giannis Dekoulakos - p2822110

The report consists of 2 sections, the first section presents a data analysis using REDIS and the second one using MongoDB.

The programming language used is R. The CLI was also used to run queries against the MongoDB

Task1

0. Setup

- We set up a docker container with redis on it.
- We read the data from the source.
- Insection of the data
- We connect to the redis instance.

```
curl -sSL https://raw.githubusercontent.com/bitnami/bitnami-docker-
redis/master/docker-compose.yml > docker-compose.redis.yml
docker-compose -f docker-compose.redis.yml up -d
```

```
### Read the data

modified_listings <-
read.csv("D:\\Downloads\\RECORDED_ACTIONS\\modified_listings.csv")
emails_sent <- read.csv("D:\\Downloads\\RECORDED_ACTIONS\\emails_sent.csv")

str(modified_listings)
str(emails_sent)
```

```
### connection to redis
# install.packages("redux")

# Load the library
library("redux")

# Create a connection to the local instance of REDIS
r <- redux::hiredis(
  redux::redis_config(
    host = "127.0.0.1",
    port = "6379"))
```

```
r$FLUSHALL()
```

1. How many users modified their listing on January

```
### q1
require(dplyr)
modified_listings.month1 <- filter(modified_listings, MonthID == 1) %>%
select(UserID, ModifiedListing)
apply(modified_listings.month1, 1, function(x) r$SETBIT("ModificationsJanuary",
x[1], x[2]))

str(modified_listings.month1)

q1.res <- r$BITCOUNT("ModificationsJanuary")

r$HSET("results", "q1", q1.res)
r$HGET("results", "q1")
```

Answer: 9969 users

2. How many users did NOT modify their listing on January?

```
### q2

r$BITOP("NOT", "NonModificationsJanuary", "ModificationsJanuary")
q2.res <- r$BITCOUNT("NonModificationsJanuary")

r$HSET("results", "q2", q2.res)
r$HGET("results", "q2")

# total users count through REDIS: 20000
r$BITOP("OR", "TotalUsers", c("NonModificationsJanuary", "ModificationsJanuary"))
total_users_sum <- r$BITCOUNT("TotalUsers")

# total users count in the dataset: 19999
total_users_real <- count(modified_listings.month1)
```

Answer: 10031 users

The reason that the number of total user (19999) is now equal the summary of results of the questions 1.1 and 1.2 (20000) is that used bitmaps that increment in byte (8 bit) level and the number 19999 is not a divisor of 8, while the next number 20000 is. So, the function BITOP used 2500 bytes (20000 bits = 20000 users) to cover the need of 19999 users.

3. How many users received at least one e-mail per month (at least one e-mail in January and at least one e-mail in February and at least one e-mail in March)?

```
### q3

process_month_emails <- function(month_id, seq_name, func){
  emails_sent.month <- filter(emails_sent, MonthID == month_id) %>%
    select(UserID, EmailOpened) %>%
    group_by(UserID)  %>%
    summarize(EmailOpened=sum(EmailOpened)) %>%
    apply(1, function(x) func(x, seq_name))
}

set_email_sent <- function(x, seq_name) r$SETBIT(seq_name, x[1], 1)

process_month_emails(1, "EmailsJanuary", set_email_sent)
process_month_emails(2, "EmailsFebruary", set_email_sent)
process_month_emails(3, "EmailsMarch", set_email_sent)

r$BITOP("AND", "ReceivedEmail", c("EmailsJanuary", "EmailsFebruary",
"EmailsMarch"))

q3.res <- r$BITCOUNT("ReceivedEmail")
r$HSET("results", "q3", q3.res)
r$HGET("results", "q3")
```

Answer: 2668 users

4. How many users received an e-mail on January and March but NOT on February?

```
### Q4

r$BITOP("NOT", "EmailsFebruaryInverted", "EmailsFebruary")
r$BITOP("AND", "ReceivedEmail:Q4", c("EmailsJanuary", "EmailsFebruaryInverted",
"EmailsMarch"))

q4.res <- r$BITCOUNT("ReceivedEmail:Q4")
r$HSET("results", "q4", q4.res)
r$HGET("results", "q4")
```

Answer: 2417 users

5. How many users received an e-mail on January that they did not open but they updated their listing anyway?

```
### q5

mark_has_not_opened_mail <- function(x, seq_name){
  bit <- ifelse(x[2]>0, 0, 1)
  r$SETBIT(seq_name, x[1], bit)
}
```

```
process_month_emails(1, "EmailsNotReadJanuary", mark_has_not_opened_mail)

r$BITOP("AND", "EmailsNotOpenedAndModifiedJanuary", c("EmailsNotReadJanuary",
"ModificationsJanuary"))

q5.res <- r$BITCOUNT("EmailsNotOpenedAndModifiedJanuary")
r$HSET("results", "q5", q5.res)
r$HGET("results", "q5")
```

Answer: 1961 users

6. How many users received an e-mail on January that they did not open but they updated their listing anyway on January OR they received an e-mail on February that they did not open but they updated their listing anyway on February OR they received an e-mail on March that they did not open but they updated their listing anyway on March?

```
### q6
process_month_emails(2, "EmailsNotReadFebruary", mark_has_not_opened_mail)
process_month_emails(3, "EmailsNotReadMarch", mark_has_not_opened_mail)

modified_listings.month2 <-
  filter(modified_listings, MonthID == 2) %>% select(UserID, ModifiedListing) %>%
  apply(1, function(x) r$SETBIT("ModificationsFebruary", x[1], x[2]))

modified_listings.month3 <-
  filter(modified_listings, MonthID == 3) %>% select(UserID, ModifiedListing) %>%
  apply(1, function(x) r$SETBIT("ModificationsMarch", x[1], x[2]))

r$BITOP("AND", "NotOpenedAndModifiedFebruary", c("EmailsNotReadFebruary",
"ModificationsFebruary"))
r$BITOP("AND", "NotOpenedAndModifiedMarch", c("EmailsNotReadMarch",
"ModificationsMarch"))

r$BITOP("OR", "NotReceivedModified", c("EmailsNotOpenedAndModifiedJanuary",
"NotOpenedAndModifiedFebruary", "NotOpenedAndModifiedMarch"))

q6.res <- r$BITCOUNT("NotReceivedModified")
r$HSET("results", "q6", q6.res)
r$HGET("results", "q6")
```

Answer: 5249 users

7. Does it make any sense to keep sending e-mails with recommendations to sellers? Does this strategy really work? How would you describe this in terms a business person would understand?

```
### q7

not_opened_modified.January <- r$BITCOUNT("EmailsNotOpenedAndModifiedJanuary")
not_opened_modified.February <-r$BITCOUNT("NotOpenedAndModifiedFebruary")
not_opened_modified.March <- r$BITCOUNT("NotOpenedAndModifiedMarch")

##total user not opened and modified

total_not_opened_modified <- not_opened_modified.January +
not_opened_modified.February + not_opened_modified.March

##total user opened and modified

##january

set_email_opened <- function(x, seq_name) {
  bit <- ifelse(x[2]>0, 1, 0)
  r$SETBIT(seq_name, x[1], bit)
}

process_month_emails(1, "EmailsOpenedJanuary", set_email_opened)

total_emails_opened_January <- r$BITCOUNT("EmailsOpenedJanuary")

r$BITOP("AND", "EmailsOpenedAndModifiedJanuary", c("EmailsOpenedJanuary",
"ModificationsJanuary"))
jan_opened_modified <- r$BITCOUNT("EmailsOpenedAndModifiedJanuary")

##feb
process_month_emails(2, "EmailsOpenedFebruary", set_email_opened)

r$BITOP("AND", "EmailsOpenedAndModifiedFebruary", c("EmailsOpenedFebruary",
"ModificationsFebruary"))
feb_opened_modified <- r$BITCOUNT("EmailsOpenedAndModifiedFebruary")

##March

process_month_emails(3, "EmailsOpenedMarch", set_email_opened)

r$BITOP("AND", "EmailsOpenedAndModifiedMarch", c("EmailsOpenedMarch",
"ModificationsMarch"))
March_opened_modified <- r$BITCOUNT("EmailsOpenedAndModifiedMarch")

##total user opened and modified
total_opened_modified <-March_opened_modified + feb_opened_modified +
jan_opened_modified
total_opened_modified > total_not_opened_modified # True

# total_opened_modified > total_not_opened_modified ara to na stelnoun email
epireazei to modification na synexisoun

# posoi user exoun kanei modified
user <- subset(modified_listings, modified_listings$ModifiedListing == "1")
```

```
user_uniq <- length(unique(user$UserID))
total_user <- q1.res + q2.res

user_uniq/total_user
# user_uniq/total_user = 17541/20000 =0.877 exoun kanei modified ara ena 12% den
exei kanei modified ayth tha einai kai h diafora sta pososta pio katw

# total unique user
total_unique_user <- length(unique(emails_sent$UserID))

#percentage of user who modified and opened the emails

percentage_opened <- paste((total_opened_modified / total_unique_user)*100,"%")

# "52.8176933649881 %"

# percentage of  user who  modified but they did NOT open the emails

percentage_not_opened <- paste((total_not_opened_modified /
total_unique_user)*100,"%")

# "36.8486817443459 %"

percentage_opened > percentage_not_opened

#to pososto user  apo to modified email einai megalutero ekeinwn poy exoun
anoixtoi apo tous user ara na synexizoun na stelnoun
#52.8176933649881% opened kai modified
#36.8486817443459% NOTopened kai modified
#12% NOT modified genika
```

So, it seems that the emails affect the modification, so we suggest they keep sending emails with recommendation to sellers. This strategy of sending emails to the customers can lead to more modifications

#52.8176933649881% opened kai modified #36.8486817443459% NOTopened kai modified #12% NOT modified general

Task 2

Question 1

We setup a docker container to host mongodb.

```
curl -sSL https://raw.githubusercontent.com/bitnami/bitnami-docker-
mongodb/master/docker-compose.yml > docker-compose.mongo.yml

docker-compose -f docker-compose.mongo.yml up -d
```

Install and import the required packages.

```
### import library and create connection

# Install the mongolite driver
install.packages("mongolite")

# Load mongolite
library("mongolite")

# Install the jsonlite package
install.packages("jsonlite")

# Load jsonlite
library("jsonlite")
```

Create the database and its collection:

```
# not really needed, first connection with R will create this.
use big_data_systems
db.createCollection('data')
```

Open a connection and create an object that 'holds' the connection info.

```
m <- mongo(collection = "data", db = "big_data_systems", url =
"mongodb://localhost:27017")
```

Load the JSON file paths in a text file. 1 path per line.

Note: We are using a slightly different version of the command given in the instruction. We are accessing the FullName of the file, so that we will not need to set working directory in R.

```
### Load json files index.
## windows
# Getting the full path, so setwd() will not be required.
dir ./BIKES -R -file | ForEach-Object {$_.FullName} > files_list.txt
```

Read the index from the file in a vector.

```
files_index <- readLines("D:\\Downloads\\BIKES_DATASET\\files_list.txt",
encoding="UTF-8")
```

We wrote a function that reads a JSON file as an R object and cleans it.

Specifically, we removed properties `query` and `ad_seller`. Then, we tried to clean the `Price` field of the `ad_data` property. We replaced the euro sign(`\u20ac` in unicode), the reversed dots and the spaces. By doing oll that we ensured that we will end up with a numeric string in the field. At that point we were ready to convert it to numeric.

We are now ready to transform the JSON back to text and import it, using the libraries' code.

We then apply the function for each filepath in the index.

In this code block, we have the drop function as a reference.

```
# Testing: path <- "D:\\Downloads\\Redis-Mongo Assignment-v1.3\\Redis-Mongo
Assignment-v1.3\\tmp\\20156358.json"
clean_and_import_data <- function (path){
  json_data <- read_json(path, encoding = "utf-8")

  # Remove not needed data
  json_data$query <- NULL
  json_data$ad_seller <- NULL

  # clean price
  json_data$ad_data$Price <- gsub("\u20ac", "", json_data$ad_data$Price)
  json_data$ad_data$Price <- gsub("\\.", "", json_data$ad_data$Price)
  json_data$ad_data$Price <- gsub(" ", "", json_data$ad_data$Price)

  json_data$ad_data$Price <- as.numeric(json_data$ad_data$Price)

  # Convert the r object back to JSON format
  json_data <- toJSON(json_data, auto_unbox = TRUE)

  # insert the converted object
  m$insert(json_data)
}

# foreach filepath, execute the function.
for(path in files_index){
  clean_and_import_data(path)
}

## drop data
m$drop()
```

Question 2

Find the count of the documents in the collection.

Answer: 29701

```
db.data.count()
// 29701
```

Question 3

Find the average price in the dataset.

We used an aggregation to find the answer: 2962.7

We notice that there are some documents that have stored Price as a non-numeric value. We found them using the filter `{ "ad_data.Price": { $type: "int" } }`. All of them seem to be tagged as "NA". A bit of debugging showed that those NAs came from the documents that had the `Askforprice` value in the price field and transformed to NA by R.

Used for calculation: (only the numerics) 29145

```
db.data.aggregate({"$group":{"_id":null, "avg_price":{"$avg": "$ad_data.Price"}}})
//{ "_id" : null, "avg_price" : 2962.701012180477 }

// used for calc
db.data.find({"ad_data.Price": {$ne:"NA"}}).count()
// 29145

// A more appropriate aggregation
db.data.aggregate([{$match: {"ad_data.Price": {"$ne":"NA"}}}, {"$group":
{"_id":null, "avg_price":{"$avg": "$ad_data.Price"}}}])
// this will return the same result
```

Question 4

Next, we are trying to find the minimum and the maximum price in the dataset.

The maximum value is found to be 89000, with a simple filtering and sorting of the data.

The minimum value was found to be 1, which is not valid. A lot of ads are using 1, 10, etc. to show that the price is not defined. We are going to filter the results. We remove the 0.5 % of the values after sorting the dataset by price.

The minimum value now is 140, which makes sense for motors(used).

```
// Maximum
db.data.find({"ad_data.Price": {$ne:"NA"}}, {"ad_data.Price":
1}).sort({"ad_data.Price":-1}).limit(1)
// { "_id" : ObjectId("61f4cbbb821f00006a040791"), "ad_data" : { "Price" : 89000 }
}

// Minimum
var count = db.data.find({"ad_data.Price": {$ne:"NA"}}).count()
var limit = Math.ceil(.995*count)
db.data.aggregate([   {$sort:{"ad_data.Price":-1}},   {$limit:limit},   {$group:
{"_id":null,"min":{$min:"$ad_data.Price"}}} ])
```

```
// { "_id" : null, "min" : 140 }
```

Question 5

We now filter the values by any valid translation of 'Negotiable' in greek. That will give us the number 1849.

In the process, we created an index on the `description` field and on the `metadata.model` (because we were testing the search in various fields and we saw that `metadata.model` includes the keyword negotiable) with type of text, so that it will include case and diacritic insensitive variations of the list of words we gave it.

```
db.data.createIndex({description: "text", "metadata.model": "text"})

var negotiable_greek_keywords = ["διαπραγματευσιμη", "συζητησιμη", "negotiable"]

var search = negotiable_greek_keywords.join(" ")

db.data.find({"$text": {"$search": search}}).count()
// 1849
```