Civil Service Web Application

MPCS 53014 Big Data Application Architecture

Final Project

Gonzalo Oyanedel Vial (gov@chicagobooth.edu)

Table of contents

The Data

The bulk of the data was downloaded from the National Electoral Service of Chile (SERVEL). Original data can be found here.

Given that the data was anonymized, we had to re-create fictious information: full names, citizen's id, voting centers and voting units. This was performed using the following code in R:

```
library('openxlsx')
library('randomNames')
library("digest")

data <- read.csv("VW_VOTARON_2020PLEB_Datos completos.csv", sep=";", fileEncoding
= 'UTF-8-BOM')

str(data)

data$unique_id = seq(1:nrow(data))
data$full_name = randomNames(nrow(data),
                    ethnicity=4,
                    which.names="both",
                    name.order="last.first",
                    name.sep=", ",
                    sample.with.replacement=TRUE,
                    return.complete.data=FALSE)

digg <- function(x) {
  return(digest(x, "md5", serialize = FALSE))
      }

data$digest = sapply(paste(data$unique_id, data$full_name), digg)

data <- data[order(data$Circunscripcion),]
```

```
circunscrip = unique(data$Circunscripcion)

collector = data[1,]
collector$Mesa = 1
collector$Local = 1
collector = collector[FALSE,]; collector

for (i in 1:length(circunscrip)){
    temp = data[data$Circunscripcion == circunscrip[i],]
    N = nrow(temp)
    mesas = ceiling(N/200)
    temp$Mesa = sample(1:mesas,N, replace=T)
    locales = ceiling(N/10000)
    temp$Local = sample(1:locales, N, replace=T)
    collector = rbind(collector, temp)
}

write.csv(collector, "enriched data/enriched_electoral_data.csv")
```

As it can be seen in the code above, we used the library `randomNames` to create names for the almost 15 million anonymized citizens. Additionally, voting centers (locations) and units (specific voting booths within the locations) were created using historical statistics. An additional minor step to simplify the database was performed to solve formatting inconsistencies.

After exporting the database as `csv`, this was uploaded to the server.

On the next sections, we will go through the process of creation of the web application with focus on the big data technologies.

Citizens' module

First, we create the database in `hive`:

```
hive> CREATE TABLE gov_electoral_data (X STRING, Circunscripcion STRING, Edad INT,
Nacionalidad STRING, Partido STRING, Provincia STRING, Region STRING, Sexo STRING,
Sufragio STRING, VotoExterior STRING, Numero_de_registros INT, Votaron INT,
unique_id INT, full_name STRING, digest STRING, Mesa INT, Locale INT) row format
delimited fields terminated by ',';
```

Then, after deploying the database to the server, we copy the data from the `.csv` file into Hive:

```
hive> LOAD DATA LOCAL INPATH
'/home/hadoop/gov/civil_service_data/data/enriched_electoral_data_simplified.csv'
OVERWRITE INTO TABLE gov_electoral_data;

hive> create table gov_electoral_data_2 as select unique_id, X, Circunscripcion,
Edad, Nacionalidad, Partido, Provincia, Region, Sexo, Sufragio, VotoExterior,
Numero_de_registros, Votaron, full_name, digest, Mesa, Locale from
```

```
gov_electoral_data where unique_id is not null

hive> describe gov_electoral_data;

hbase> create 'gov_electoral_data', 'elect'

hive>
create external table gov_electoral_data_hbase
    (unique_id BIGINT, x STRING, circunscripcion STRING, edad BIGINT, nacionalidad
STRING, partido STRING, provincia STRING, region STRING, sexo STRING, sufragio
STRING, votoexterior STRING, numero_de_registros BIGINT, votaron BIGINT, full_name
STRING, digest STRING, mesa BIGINT, locale BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' =
':key,elect:x,elect:circunscripcion,elect:edad,elect:nacionalidad,elect:partido,el
ect:provincia,elect:region,elect:sexo,elect:sufragio,elect:votoexterior,elect:nume
ro_de_registros,elect:votaron,elect:full_name,elect:digest,elect:mesa,elect:locale
')
TBLPROPERTIES ('hbase.table.name' = 'gov_electoral_data');

hive> insert overwrite table gov_electoral_data_hbase select * from
gov_electoral_data_2;
```

We can verify our database is in `hbase` by running:

```
hbase> scan 'gov_electoral_data'
```

Then, we upload all `html`, `mustache` and `js` files to the webserver. In a SSH connection under the application folder, we run the below code to check the application is working properly:

```
node app.js YYYY QQQ.WW.ZZ.XX 8070
```

Stats module

For the second module we follow a similar process, however, in this case there are some additional `spark` steps as we have to process some data to create the aggregated views.

Additionally, we uploaded information for every district in terms of poverty to enrich our district statistics. This data is merged with our current individual data.

```
hive> CREATE TABLE gov_circunscripcion_enrichment (Circunscripcion STRING,
poverty_income FLOAT, poverty_multi FLOAT, vulnerability STRING) row format
delimited fields terminated by ';';
hive> LOAD DATA LOCAL INPATH
'/home/hadoop/gov/civil_service_data/data/circuns_2.csv' OVERWRITE INTO TABLE
gov_circunscripcion_enrichment;
```

```
hive> create table gov_circunscripcion_enrichment_2 as (select * from
gov_circunscripcion_enrichment where poverty_income is not null);
```

Then, in scala we run:

```
scala> val gov_electoral_data = spark.table("gov_electoral_data")
scala> val gov_circuns_data = spark.table("gov_circunscripcion_enrichment_2")
scala> val gov_enriched = gov_electoral_data /*Just convention*/
scala> gov_enriched.columns.toSeq

/* Electoral info*/
scala> val gov_enriched_grouped =
gov_enriched.groupBy(gov_enriched("circunscripcion")).agg(sum("votaron").as("voted
"), count("full_name").as("voters"))
scala> gov_enriched_grouped.show

/* Vulnerability info */
scala> val gov_enriched_grouped_2 =
gov_enriched_grouped.withColumn("circunscripcion_2",
expr("substring(circunscripcion, 2, length(circunscripcion)-2)"))
scala> val gov_enriched_grouped_vul =
gov_enriched_grouped_2.join(gov_circuns_data,
gov_enriched_grouped_2("circunscripcion_2") <=>
gov_circuns_data("circunscripcion"),
"left").drop(gov_circuns_data("circunscripcion"))
scala> gov_enriched_grouped_vul.show

/* Saving to hive*/
scala> import org.apache.spark.sql.SaveMode
scala>
gov_enriched_grouped_vul.write.mode(SaveMode.Overwrite).saveAsTable("gov_electoral
_stats")
```

Then moving data to hbase through hive:

```
hbase> create 'gov_electoral_stats', 'stat'

hive>
create table gov_electoral_stats_2 as select circunscripcion_2, voters, voted,
poverty_income, poverty_multi, vulnerability from gov_electoral_stats;
hive> drop table gov_electoral_stats;
hive> create table gov_electoral_stats as select * from gov_electoral_stats_2;
hive> create external table gov_electoral_stats_hbase
    (circunscripcion STRING, voted BIGINT, voters  BIGINT, poverty_income FLOAT,
poverty_multi FLOAT, vulnerability STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' =
':key,stat:voted,stat:voters,stat:poverty_income,stat:poverty_multi,stat:vulnerabi
lity')
```

```
TBLPROPERTIES ('hbase.table.name' = 'gov_electoral_stats');

hive> insert overwrite table gov_electoral_stats_hbase select * from
gov_electoral_stats;
```

Finally, with the data stored in `hive`, we run `node app.js YYYY QQQ.WW.ZZ.XX 8070` to check the applicatio is working properly.

Deployment

Deployment of civil service application using CodeDeloy can be found here.

The application files were stored in a `zip` folder and uploaded to S3 at AWS. These files can be found at `s3://mpcs53014-gov/app_civil_service.zip`. Then, a classic Load Balancer was set up. The Load Balancer handles the requests to the application on the side of the front end. This, added to the big data architecture using `hbase` in the backend, makes the application resilient to a massive amount of requests from users.

In real life this application would be used by around 6 million citizens in a few dats that need to check their voting information before every election, thus the proposed architecture would be appropiate to manage such traffic.

Deployment information

- Application: `gov-civil-service`
- Deployment ID: `d-70M5BXDFD`
- Deployment group: `gov-production-deployment-group-civil-service`
- Load balancer: `gov-loadbalancer-civil-service`

Other considerations

Three `html` files were designed to provide a simple navigation experience. The first one is the `index.html` that allows the user move to wether the citizen module or the stats module. The other two correspond to `stats.html` and `citizen.html` and have the same structure: a simple `html` format connected via `iframe` to `mustache` files serving as template to show the information obtained from the `hbase` tables.

The code for the `html`, `css` and `mustache` can be found in the temporary repository set up for this project.