Implementacion de una arquitectura de *Big Data* usando diversos frameworks y lenguajes
Métodos de Gran Escala - Proyecto Final

Fernanda Mora - Andrea García - Sergio Haro
Alfonso Kim - Luis Román

Junio 5, 2015

1 Introducción

Para el proyecto final de la materia de Gran Escala se utilizaron dos bases de datos. La primera contiene la información referente al número de avistamientos de OVNIS en los Estados Unidos desde 1900 hasta la fecha. Se tienen variables como Estado en donde se hizo el avistamiento, forma del avistamiento, duración del avistamiento, entre otras.

La segunda base se extrajo del proyecto "Global Data on Events, Location and Tone (GDELT)" que contiene más de 300 millones de eventos geolocalizados desde 1979 a la fecha. El proyecto de GDELT extrae noticias en más de 100 idiomas e identifica a las personas, lugares, organizaciones involucradas en diferentes eventos. En esta base se tienen variables relativas al evento y a los actores involucrados, como por ejemplo fecha de ocurrencia del evento, quién fue el actor que inició el evento, quién lo recibió, en dónde, qué importancia tuvo el evento, etc.

El objetivo de este proyecto es practicar las herramientas vistas en clase tales como: Sqoop, Flume, Hive o Impala, Luigi y Pig o PySpark.

Este trabajo no abarca todas las herramientas para analítica vistas en clase (Pig, MapReduce, etc) ya que consideramos algunas son repetitivas y decidimos utilizar las que mejor se adaptaran a las necesidades del proyecto. Cabe mencionar que en la fase de ingesta sí comparamos Scoop vs Kite y Flume.

El proyecto está basado en las notas de clase del Dr. De Unánue: `https://github.com/ITAM-DS/big-data/blob/master/lecture_4/lecture_4.org`

2 Arquitectura

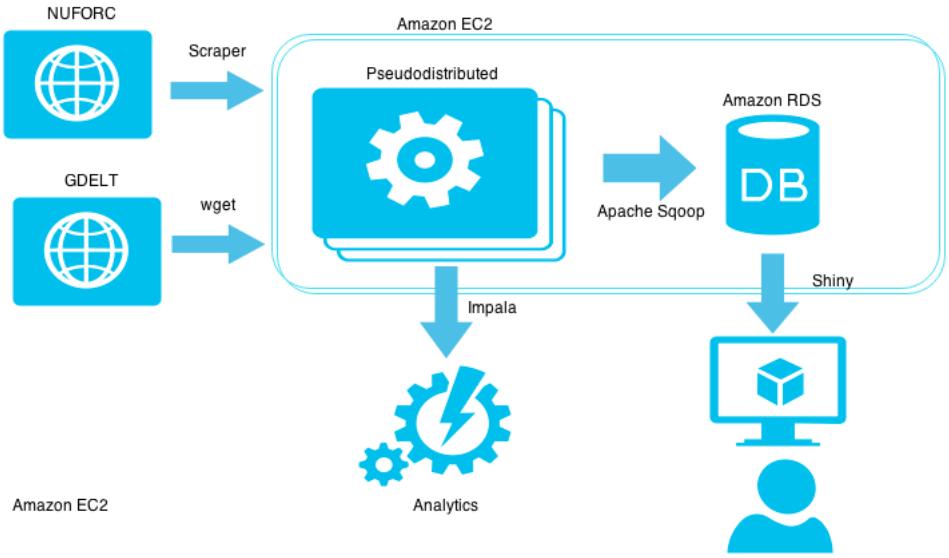
Para la elaboración del proyecto se usó una máquina virtual en Amazon EC2 del tipo m3.large (2 CPUs virtuales y 7.5 G de RAM), se construyó a partir de

la definición de contenedor Dockerfile de nanounanue/docker-hadoop visto en clase.

Para el diseño de flujo de datos y su orquestación se usó Luigi, a continuación se describe el flujo de datos completo. Cabe resaltar que no se realizaron los pasos 4 y 5 ya que no era el objetivo de la clase. Esa parte se vio en el taller de Visualización que impartieron a mediados de semestre y en la materia de Aprendizaje de máquina.

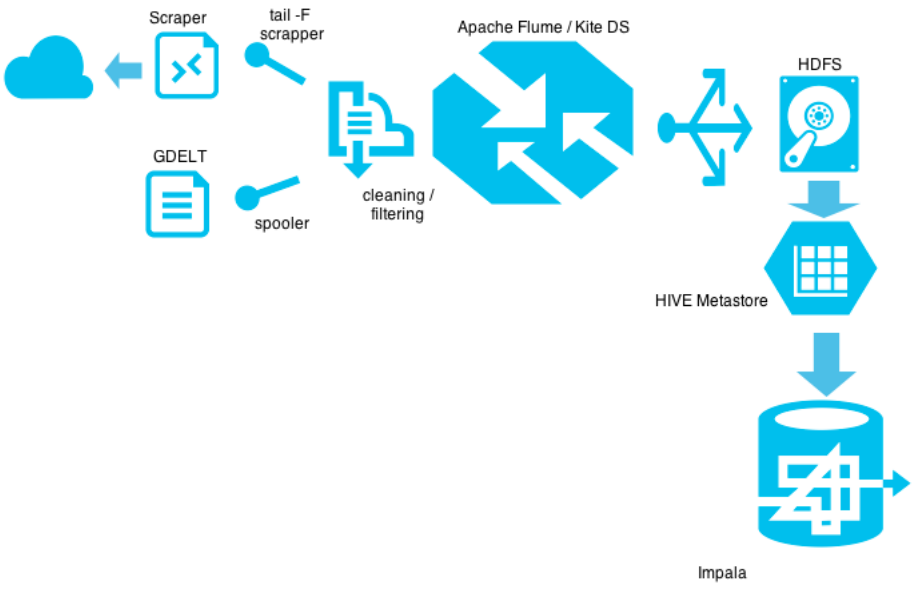
1. Recuperar
    - Verificar el formato de los datos
    - Preparar el repositorio de Información Obtención de datos : la base de UFO se obtuvo con un scraper mientras que la de GDELT se obtuvo con wget
2. Limpieza de datos
    - Análisis exploratorio de datos
    - Definir el método de limpieza y normalización de datos
    - Realizar limpieza de datos, en este caso se eliminaron las observaciones incompletas y las que estaban fuera del área geográfica (EUA en el caso de UFO)
    - Persistir resultados en nuevo repositorio
3. Carga
    - Definir estructuras procesamiento líneas de operación
    - Diseñar estructura de DWH's
    - Procesar información y re procesarla, dependiendo de las necesidades del flujo de información
    - Cargar información a DWH's
4. Analítica y Aprendizaje Máquina
    - Ejecución de algoritmos de aprendizaje de máquina dependiendo de las necesidades del cliente.
5. Visualización de resultados: en esta parte se pueden usar herramientas como gráficas interactivas, Shiny o Gephy

A continuación se presenta el diagrama general del flujo de trabajo , seguido uno más detallado:



2.1 Ingesta

Para esta sección se comparó Apache Flume vs Kite DS. Apache Flume es mejor para automatizar la carga de los datos debido a que es un sistema de alto desempeño para la ingesta de datos:



2.2 Obtención de datos

- **UFO**
  La descarga de datos de NUFORC se realizó con el proyecto de medio semestre para hacer scraping de web con 2 pequeñas modificaciones: preformatear las fechas para que no escriba las diagonales como separador y para que el texto de salida sea usando la codificación UTF-8.

  Usando Apache Flume se configuró un proceso que escuche el final del archivo donde se descargan los datos de UFO (tail -f), mediante una fuente de tipo exec:

```
1 UFOAgent.sources.UFOScraper.type = exec
2 UFOAgent.sources.UFOScraper.command = tail -F \
  ↪ /home/itam/scrapper/nuforc/data.csv} \\
```

- **GDELT**
  Se usó la lista de archivos del proyecto de medio término, descargando sólo los últimos 10 archivos para probar.

  De igual forma se usó Apache Flume para escuchar el directorio de descarga de archivos con una fuente de tipo spool.

  Ambas fuentes de datos se distribuían a 2 sumideros distintos: HDFS y Kite.

2.3 Procesamiento

- *Sqoop*
  Se configuró una pequeña base de datos en Postgres para el volcado de datos relacionales para su posterior procesamiento por una herramienta de visualización, en este caso por Shiny.

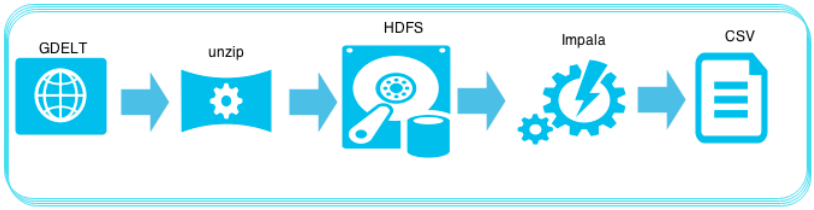
- *Impala*
  En lugar de Hive, se prefirió utilizar Impala por su rapidez. Al final del documento se encuentran las consultas que se realizaron con IMPALA.

- *Luigi*
  Luigi nos permitió implementar de manera rápida y sencilla el pipeline para los datos. Independientemente de las características atractivas de Luigi (Python, independiente o dependiente de hadoop, Python, soporte a procesos en paralelo, Python, etc.) Se tomó en cuenta que el lenguaje para crear las las tareas es común entre el equipo. Algunas de las tareas, que pudimos resolver utilizando Luigi:

  – En los ejercicios realizados de carga de GDELT a Postgres, se tenía que estar monitoreando si el archivo cargaba bien, o se detenía por algún error o falta de limpieza en la información. En caso de fallar, se tenía que borrar los registros cargados y volver a cargar el archivo (previa limpieza o corrección), por lo que esto truncaba cualquier

proceso posterior.

Flujo implementado en Luigi:



2.4 Problemas encontrados

- *Scraper*
  Para integrar el scraper de NUFORC con Apache Flume fue necesario modificar el spider para cambiar la codificación y el separador de fechas. Sin estas modificaciones apache Flume intentaba insertar caracteres inválidos en la BD (problema de codificación) o el HDFS no recibía los datos porque no acepta diagonales.

- *Sqoop*
  Fue necesario probar varios formatos en el sumidero de Flume, ya que en la configuración inicial arrojaba el siguiente error al escribir en la base de datos: "java.lang.ClassCastException: org.apache.hadoop.io.Text cannot be cast to com.cloudera.sqoop.lib.SqoopRecord"

- *Apache Hive*
  A pesar de que se reconstruyó la máquina de AWS usando el Dockerfile de la clase, algunas de las configuraciones no persistieron correctamente y al reiniciar el equipo se perdieron, específicamente la configuración de MySQL para el datastore de Hive, por lo que no se podía hacer procesamiento en Impala.

- *Luigi*
  Por problemas de red iniciamos Luigi en la computadora de Sergio y luego lo conectamos a AWS. También no reconocía unas librerias. El error que sale es : AttributeError: 'Module' object has no attribute 'Task'.

3 Conclusiones

Este proyecto nos resultó muy ilustrativo para entender las herramientas necesarias en la implementación de un arquitectura básica para un proyecto de gran escala. Incluyó desde la adquisición de datos de fuentes heterogéneas, distribuir los datos en una infraestructura de alta disonibilidad hasta hacer analítica y

reporteo con los datos procesados.

El flujo de datos debe estar orientado a procesos y no tareas fragmentadas. El proceso de orquestación es vital para asegurar la resiliencia del flujo de datos. Luigi es una excelente herramienta para realizar esto ya que es fácil de usar y a diferencia de las herramientas tradicionales de Hadoop no es necesario estar en el ambiente de Hadoop. Además Luigi te permite identificar en qué proceso tienes problemas o incluso frenar el proceso si trae errores.

En cuanto a la ingesta de datos preferimos Flume a Kite ya que Flume te permite procesar y filtrar directamente sin pasar por tantos procesos como en Kite (para leer de diferentes fuentes y esquemas, procesarlos y pasarlos a diferentes destinos etc..). Sqoop nos gustó por qué nos permite manejar los datos desde y hacia una base relacional. a pesar de ser menos potente que Flume (por qué solo es para BD relacionales) nos gustó por su facilidad de uso (se conecta a Postgres y a otras bases de datos). En el Anexo incluimos los scripts que se utilizaron para la realización de este proyecto.

4 Trabajo Futuro

Creemos que lo siguiente es implementar una arquitectura más grande en un ambiente realmente distribuido -es decir- usando muchos servidores con responsabilidades distintas; por ahora todo está contenido en la misma máquina y no se perciben los problemas del cómputo distribuido (tolerancia a fallos, latencia de red, comunicación, etc).

También sería interesante automatizar el despliegue de los scripts necesarios para montar la arquitectura, ya que para este proyecto se hizo totalmente a mano. Se podría hacer desde el Dockerfile como en clase o usando herramientas de aprovisionamiento de servidores como Chef.

5 Apéndice: Scripts

- **Flume para GDELT**

```
# Componentes
GDELTAgent.sources = GDELTSpooler
GDELTAgent.channels = archivo
#GDELTAgent.sinks = GDELTKiteDS
GDELTAgent.sinks = GDELTHDFS

# Canal
GDELTAgent.channels.archivo.type = file
GDELTAgent.channels.archivo.checkpointDir =
↪  /opt/gdelt/log/checkpoint/
GDELTAgent.channels.archivo.dataDirs = /opt/gdelt/log/data/

# Fuente e Interceptores
GDELTAgent.sources.GDELTSpooler.type = spooldir
GDELTAgent.sources.GDELTSpooler.spoolDir = /home/itam/gdelt/
GDELTAgent.sources.GDELTSpooler.channels = archivo

# Interceptor
GDELTAgent.sources.GDELTScraper.interceptors.attach-schema.type
↪  = static
GDELTAgent.sources.GDELTScraper.interceptors.attach-schema.key
↪  = flume.avro.schema.url
GDELTAgent.sources.GDELTScraper.interceptors.attach-schema.value
↪  = file:/home/itam/schemas/gdelt.avsc

# Sumidero
GDELTAgent.sinks.GDELTHDFS.type=hdfs
GDELTAgent.sinks.GDELTHDFS.channel=archivo
GDELTAgent.sinks.GDELTHDFS.hdfs.path =
↪  /user/itam/gdelt_flume/
#GDELTAgent.sinks.GDELTHDFS.hdfs.fileType = DataStream
GDELTAgent.sinks.GDELTHDFS.hdfs.fileType = SequenceFile
GDELTAgent.sinks.GDELTHDFS.hdfs.filePrefix = GDELTData

GDELTAgent.sinks.GDELTHDFS.hdfs.writeFormat = Text
GDELTAgent.sinks.GDELTHDFS.hdfs.batchSize = 1000
GDELTAgent.sinks.GDELTHDFS.hdfs.rollSize = 0
GDELTAgent.sinks.GDELTHDFS.hdfs.rollCount = 10000
```

- **Flume para UFO**

```
# Componentes
UFOAgent.sources = UFOScraper
```

```
3  UFOAgent.channels = archivo
4  #UFOAgent.sinks = UFOKiteDS
5  UFOAgent.sinks = UFOHDFS
6
7  # Canal
8  UFOAgent.channels.archivo.type = file
9  UFOAgent.channels.archivo.checkpointDir =
   ↪  /opt/ufos/log/checkpoint/
10 UFOAgent.channels.archivo.dataDirs = /opt/ufos/log/data/
11
12 # Fuente e Interceptores
13 UFOAgent.sources.UFOScraper.type = exec
14 UFOAgent.sources.UFOScraper.command = tail -F
   ↪  /home/itam/scrapper/nuforc/data.csv
15 UFOAgent.sources.UFOScraper.channels = archivo
16
17 # Interceptor
18 UFOAgent.sources.UFOScraper.interceptors = attach-schema
   ↪  morphline
19
20 UFOAgent.sources.UFOScraper.interceptors.attach-schema.type
   ↪  = static
21 UFOAgent.sources.UFOScraper.interceptors.attach-schema.key =
   ↪  flume.avro.schema.url
22 UFOAgent.sources.UFOScraper.interceptors.attach-schema.value
   ↪  = file:/home/itam/schemas/ufos.avsc
23
24 UFOAgent.sources.UFOScraper.interceptors.morphline.type =
   ↪  org.apache.flume.sink.solr.morphline.MorphlineInterceptor$Builder
25 UFOAgent.sources.UFOScraper.interceptors.morphline.morphlineFile
   ↪  = /home/itam/ingesta/morphline.conf
26 UFOAgent.sources.UFOScraper.interceptors.morphline.morphlineId
   ↪  = convertUFOFileToAvro
27
28 # Sumidero
29 UFOAgent.sinks.UFOHDFS.type=hdfs
30 UFOAgent.sinks.UFOHDFS.channel=archivo
31 UFOAgent.sinks.UFOHDFS.hdfs.path = /user/itam/ufos_flume/
32 #UFOAgent.sinks.UFOHDFS.hdfs.fileType = DataStream
33 UFOAgent.sinks.UFOHDFS.hdfs.fileType = SequenceFile
34 UFOAgent.sinks.UFOHDFS.hdfs.filePrefix = UFOData
35 UFOAgent.sinks.UFOHDFS.hdfs.writeFormat = Text
36 UFOAgent.sinks.UFOHDFS.hdfs.batchSize = 1000
37
38 UFOAgent.sinks.UFOHDFS.hdfs.rollSize = 0
39 UFOAgent.sinks.UFOHDFS.hdfs.rollCount = 10000
```

```
40
41 # Sumidero KITE
42 UFOAgent.sinks.UFOKiteDS.type =
   ↪ org.apache.flume.sink.kite.DatasetSink
43 UFOAgent.sinks.UFOKiteDS.channel = archivo
44 UFOAgent.sinks.UFOKiteDS.kite.repo.uri =
   ↪ dataset:hive://0.0.0.0:9083/ufos
45 UFOAgent.sinks.UFOKiteDS.kite.dataset.name = ufos
46 UFOAgent.sinks.UFOKiteDS.kite.batchSize = 10
```

- **Scoop**

```
1 sqoop export \
2 --connect
   ↪ jdbc:postgresql://mge.czf1srpvl5.us-west-2.rds.amazonaws.com/mge
   ↪ \
3 --username postgres --password this_is_not_the_password \
4 --export-dir /user/itam/ufos_flume  --table ufo
5 --columns date_time \
6 --input-fields-terminated-by "|"
```

- **Impala Gdelt**
  Primero creamos la tabla. en este caso una sola partición, que lea los archivos desde la ubicación en hdfs:

```
1  DROP TABLE IF EXISTS gdelt;
2
3  CREATE EXTERNAL TABLE gdelt
4  (
5    globaleventid BIGINT,
6    sqldate STRING,
7    MonthYear STRING,
8    Year STRING,
9    FractionDate DOUBLE,
10   Actor1Code STRING,
11   Actor1Name STRING,
12   Actor1CountryCode STRING,
13   Actor1KnownGroupCode STRING,
14   Actor1EthnicCode STRING,
15   Actor1Religion1Code STRING,
16   Actor1Religion2Code STRING,
17   Actor1Type1Code STRING,
18   Actor1Type2Code STRING,
19   Actor1Type3Code STRING,
20   Actor2Code STRING,
21   Actor2Name STRING,
22   Actor2CountryCode STRING,
```

```
23    Actor2KnownGroupCode STRING,
24    Actor2EthnicCode STRING,
25    Actor2Religion1Code STRING,
26    Actor2Religion2Code STRING,
27    Actor2Type1Code STRING,
28    Actor2Type2Code STRING,
29    Actor2Type3Code STRING,
30    IsRootEvent INT,
31    EventCode STRING,
32    EventBaseCode STRING,
33    EventRootCode STRING,
34    QuadClass INT,
35    GoldsteinScale DOUBLE,
36    NumMentions INT,
37    NumSources INT,
38    NumArticles INT,
39    AvgTone DOUBLE,
40    Actor1Geo_Type INT,
41    Actor1Geo_FullName STRING,
42    Actor1Geo_CountryCode STRING,
43    Actor1Geo_ADM1Code STRING,
44    Actor1Geo_Lat FLOAT,
45    Actor1Geo_Long FLOAT,
46    Actor1Geo_FeatureID INT,
47    Actor2Geo_Type INT,
48    Actor2Geo_FullName STRING,
49    Actor2Geo_CountryCode STRING,
50    Actor2Geo_ADM1Code STRING,
51    Actor2Geo_Lat FLOAT,
52    Actor2Geo_Long FLOAT,
53    Actor2Geo_FeatureID INT,
54    ActionGeo_Type INT,
55    ActionGeo_FullName STRING,
56    ActionGeo_CountryCode STRING,
57    ActionGeo_ADM1Code STRING,
58    ActionGeo_Lat FLOAT,
59    ActionGeo_Long FLOAT,
60    ActionGeo_FeatureID INT,
61    dateadded INT,
62    sourceurl STRING
63  )
64  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES
     ↪  TERMINATED BY '\n'
65  LOCATION '/user/itam/datasets/gdelt';
```

Luego subimos los CAMEO EVENT CODES

```
DROP TABLE IF EXIST cameo_codes;
CREATE EXTERNAL TABLE cameo_codes
(
    cameoeventcode STRING,
    eventdescription string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES
 ↪  TERMINATED BY '\n'
LOCATION '/user/itam/datasets/cameo';
```

Probamos que funcione:

```
# Validamos los eventos mas recurrentes
WITH EventsMX AS (
SELECT MonthYear, EventRootCode, EventCode,
 ↪  count(IF(ActionGeo_CountryCode = 'MX',1,NULL)) c_mx
FROM GDELT
GROUP BY MonthYear, EventRootCode, EventCode
)SELECT * FROM EventsMX
WHERE c_mx >0
ORDER BY c_mx DESC LIMIT 20;

#Unimos los eventos con su descripcion.

# Validamos los eventos mas recurrentes
WITH EventsMX AS (
SELECT MonthYear, EventRootCode, EventCode,
 ↪  count(IF(ActionGeo_CountryCode = 'MX',1,NULL)) c_mx
FROM GDELT
GROUP BY MonthYear, EventRootCode, EventCode
)SELECT * FROM EventsMX em INNER JOIN cameo_codes cc ON
 ↪  em.EventCode = cc.cameoeventcode
WHERE c_mx >0
ORDER BY c_mx DESC LIMIT 20;
```

- **Luigi**

```
import luigi
import luigi.contrib
import luigi.contrib.hdfs
import luigi.contrib.hadoop

class GetFile_Task(luigi.Task):

    url =
     ↪  u'http://data.gdeltproject.org/events/20150602.export.CSV.zip'

```

```
10      def output(self):
11          import urllib
12          urllib.urlretrieve(self.url,
            ↪  '20150602.export.CSV.zip')
13          return luigi.LocalTarget('get_task.txt')
14
15
16  class Decompress_Task(luigi.Task):
17      def requires(self):
18          return GetFile_Task()
19
20      def output(self):
21          import os
22          print "Descomprimiento...\n"
23          os.system('unzip -q 20150602.export.CSV.zip')
24          return luigi.LocalTarget('20150602.export.CSV')
25
26      def run(self):
27          with self.input().open() as f:
28              print f.read()
29
30          with self.output().open('w') as f:
31              f.write('done')
32
33  class UploadHDFS_Task(luigi.Task):
34
35      def requires(self):
36          return Decompress_Task() # [Decompress_Task(date)
            ↪  for date in self.date_interval]
37
38
39      def output(self):
40          print("Moviendo archivo...\n")
41          return
            ↪  luigi.contrib.hdfs.HdfsTarget("/user/itam/datasets/gdelt/20150602.export.CS
42
43      def run(self):
44          with self.input().open() as f:
45              # process the result here
46              print f.read()
47          with self.output().open('w') as f:
48              # crate the final output
49              f.write('done')
50
51
52
```

```
53 if __name__=='__main__':
54     luigi.run()
```