Métodos de Gran Escala
Proyecto 2

Alfonso Kim
Andrea García *

Mayo 2015

1 Máquina en Amazon Web Services

Se usó un servidor en Amazon Web Services (AWS) del tipo *r3.xlarge* con las siguientes características:

- Optimizado para memoria
- 13 Elastic Compute Units (ECU)
- 30.5 GB en memoria RAM
- 8 GB en Disco Duro de estado sólido (SSD)

Se agregó un disco duro SSD adicional de 100GB.

1.1 Configuración del servidor

El disco raíz de 8 GB se dejó para el sistema operativo (Ubuntu 14.04) y el sistema de Postgres, el disco adicional de 100GB se destinó únicamente para la base de datos.

Se instaló postgres de la misma forma que en los contenedores vistos en clase.

1.1.1 Configuración del disco duro

Una vez que se hizo el cálculo aproximado del costo del disco duro (USD 0.10/GB al mes) se creó mediante la interfaz web el volumen de 100GB y el procedimiento para configurarlo en el servidor fue el siguiente:

*El proyecto se realizó en equipo para poder complementar los conocimientos de la Maestría en Computación y la Maestría en Ciencia de Datos.

```
$ lsblk
$ sudo mkfs -t ext4 /dev/xvdb
$ sudo mkdir /data
$ sudo mount /dev/xvdb /data
$ sudo echo "/dev/xvdb /data    ext4    defaults,nofail    0    2"
↪  >> /etc/fstab
```

- Linea 1: Buscar en qué dispositivo se encuentra el nuevo disco duro
- Linea 2: Formatear el disco con sistema de archivos *ext4*
- Linea 3: Crear punto de montaje
- Linea 4: Montar el disco en el punto creado en el paso anterior
- Linea 5: Persistir el montaje

Se dejó el journal y configuración por defecto del disco.

En este punto ya se puede crear la base de datos en el disco recién configurado.

2 Configuración de Postgres

2.1 Crear la base de datos

```
$ sudo mkdir /data/pg_data
$ sudo chown -R postgres:postgres /data/pg_data
$ sudo chmod -R 700 /data/pg_data
$ pg_ctl initdb -D /data/pg_data
$ pg_ctl start -D /data/pg_data
```

Se configuró el demonio de Postgres para cambiar el directorio de datos para el arranque automático.

2.2 Optimizar la base de datos

Se modificó el archivo *postgresql.conf* para aprovechar los beneficios del disco duro de estado sólido:

```
max_connections = 3
shared_buffers = 512MB
work_mem = 16MB
maintenance_work_mem = 256MB
checkpoint_segments = 64
checkpoint_timeout = 4min
checkpoint_completion_target = 0.9
```

```
random_page_cost = 2.0
effective_cache_size = 1400MB
autovacuum = off
```

Las propiedades de configuración fueron tomadas de
https://github.com/tvondra/block_benchmark/blob/master/config/postgresql-ssd-pgbench.conf y
http://dba.stackexchange.com/questions/78242/postgresql-maximize-performance-ssd

3 Configuración de la Base de Datos

3.1 Bases de datos y esquemas

Aunque lo recomendado es crear una base de datos para cada tipo de datos que se va a analizar (UFO y GDELT) y crear distintos esquemas en base de datos que representen el workflow de análisis de datos, consideramos que para el alcance del proyecto sólo era necesario crear esquemas para cada tipo de dato en sólo una base de datos:

```
CREATE database bigdata;
\connect bigdata;
CREATE schema ufo;
CREATE schema gdelt;
-- Usuarios y permisos
CREATE ROLE itam LOGIN PASSWORD 'itam';
GRANT CONNECT ON DATABASE bigdata TO itam;
GRANT USAGE ON SCHEMA ufo TO itam;
GRANT USAGE ON SCHEMA gdelt TO itam;
GRANT SELECT ON ALL TABLES IN SCHEMA ufo TO itam;
GRANT SELECT ON ALL TABLES IN SCHEMA gdelt TO itam;
```

4 Carga de Datos

4.1 Adquisición de los Datos

Para los datos de UFO se usaron los datos del scrapper del proyecto 1. Para GDELT ejecutamos los scripts en el servidor de Amazon, que tiene un ancho de banda más grande.

4.2 Limpieza de Datos

Se limpiaron los datos **antes** de cargarlos a la base. No es lo correcto ni lo más recomendable, pero la base de GDELT iba a ocupar mucho espacio en disco y el costo de tener datos duplicados es muy alto. Para un proyecto de largo alcance lo mejor es tener distintos esquemas en base de datos (o HDFS) con las distintas etapas de procesamiento de datos.

4.2.1 UFO

La limpieza de UFO fue relativamente fácil, primero se eliminaron las líneas que no tienen el número de datos correcto y se agregó la hora a las fechas que no la tienen. Para este proceso se utilizó Python.

```
def format_datetime(header=True):
    """ Lee de la entrada estandar linea por linea. Si tiene
        9 elementos entonces intenta parsear la fecha y agregarle
        la hora (00:00:00 por defecto)
        :param header: Si hay que evitar parsear la primera linea
    """
    line = sys.stdin.readline()
    bad_dates = 0
    while line:
        data = line.strip().split('|')
        date = safe_parse_date(data[1])
        if date:
            data[1] = date.strftime('%Y-%m-%d %H:%M:%S')
        else:
            data[1] = '1900-01-01 00:00:00'
            bad_dates += 1
        if len(data) == 9:
            print '|'.join(data) # Solo escribir lineas correctas
        line = sys.stdin.readline()
```

Después se creó la tabla en la base de datos y cargar el archivo:

```
create table ufo.sightings (
    line_number integer,
    date_time timestamp with time zone,
    city_str varchar,
    state_str varchar,
    summary varchar,
    report text,
    shape varchar,
    duration_str varchar,
    posted_str varchar
);

\copy ufo.sightings from '~/data/ufo_clean.txt' with header csv
↪  delimiter '|';
```

4.2.2 GDELT

El volumen de datos de GDELT es considerable: 13 GB comprimidos y aproximadamente 80GB en texto plano. Se subieron los scripts de la tarea 4 para bajar los archivos directamente al servidor en AWS:

```bash
#!/bin/bash

curl -s http://data.gdeltproject.org/events/index.html | awk 'NR < 3 { next } match($0, /<A HREF="([.\-0-9A-Za-z]*)">/, grp){ print "http://data.gdeltproject.org/events/" grp[1] }' >> list.txt

for i in $(cat list.txt); do
  wget $i
done
```

Estos archivos tienen algunas columnas que, para un análisis inicial, no son necesarias. En un proyecto de largo alcance no hay que eliminar variables.

La tabla con las columnas de interés quedó de la siguiente manera:

```
create table gdelt.events(
    GLOBALEVENTID integer,
    SQLDATE integer,
    MonthYear integer,
    Year integer,
    FractionDate double precision,
    Actor1Code varchar,
    Actor1Name varchar,
    Actor1CountryCode varchar,
    Actor2Code varchar,
    Actor2Name varchar,
    Actor2CountryCode varchar,
    EventCode varchar,
    GoldsteinScale double precision,
    AvgTone double precision,
    Actor1Geo_FullName varchar,
    Actor1Geo_CountryCode varchar,
    Actor1Geo_Lat double precision,
    Actor1Geo_Long double precision,
    Actor2Geo_FullName varchar,
    Actor2Geo_CountryCode varchar,
    Actor2Geo_Lat varchar,
    Actor2Geo_Long double precision
);
```

Particionado de tablas

Para mejorar el desempeño de la base de datos se particionó la tabla de eventos de forma anual.

```bash
#/bin/bash

```

```
for y in {1979..2015}
do
   echo "CREATE TABLE gdelt.events_y$y ( CHECK ( Year >= $y AND
    ↪  Year < $((y+1)) ) ) INHERITS ( gdelt.events );"
done
```

Este script genera una tabla con el sufijo *_yAÑO* para cada año entre 1979 y 2015, la condición CHECK asegura que no haya registros que se salgan del rango y se ejecuta de la forma `./create_partitions.sh | psql -d bigdata`

Para que las particiones funcionen correctamente es necesario crear una función que inserte cada nuevo registro de la tabla padre en su partición correspondiente, además de un trigger que capture los inserts y llame a dicha función:

```
#/bin/bash

# Funcion que evalua el anio del nuevo registro y lo inserta en la
 ↪  tabla correspondiente
echo "CREATE OR REPLACE FUNCTION events_check_range()"
echo "RETURNS TRIGGER AS \$\$"
echo "BEGIN"
ELS=""
for y in {1979..2015}
do
    echo "${ELS}IF (NEW.Year >= $y AND NEW.Year < $((y+1))) THEN"
    echo "   INSERT INTO gdelt.events_y$y VALUES (NEW.*);"
    ELS="ELS"
done
# Si algun evento cae fuera del rango de fechas se inserta a la
 ↪  tabla padre
echo "    ELSE INSERT INTO gdelt.events VALUES (NEW.*);"
echo "    END IF;"
echo "RETURN NULL;"
echo "END;"
echo "\$\$"
echo "LANGUAGE plpgsql;"

echo ""
echo ""

# Trigger que captura el insert y llama a la funcion anterior
echo "CREATE TRIGGER insert_events_trigger"
echo "    BEFORE INSERT ON gdelt.events"
echo "    FOR EACH ROW EXECUTE PROCEDURE events_check_range();"
echo ""
```

Este script se ejecuta de la misma forma que el anterior.

Para evitar usar espacio en disco se encadenó el proceso de extracción de los archivos comprimidos, filtrado de columnas y carga a la base de datos:

```
#!/bin/bash

# esta instruccion corta las lineas con las columnas que quedaron en la tabla
FILTER="cut -d$'\t' -f1-8,16-18,27,31,35,37,38,40,41,44,45,47,48"

PG_USER=postgres
PG_DATABASE=bigdata
PG_TABLE=gdelt.events

unzip -p $1 | eval $FILTER | psql -U $PG_USER -d $PG_DATABASE -c "\copy $PG_TABLE from STDIN with csv delimiter E'\t'"
```

Este script se encarga de procesar cada archivo zip, desde descomprimirlo hasta cargarlo en la base de datos. Entonces sólo queda iterar todos los archivos:

```
#!/bin/bash

for f in *.zip
do
    ./load_postgres.sh $f
done
```

Con la ejecución de este script se cargan todos los zip a la base de datos. Es necesario crear índices para los campos por los que se va a hacer el análisis:

```
#/bin/bash

for y in {1979..2015}
do
   echo "CREATE INDEX idx_events_year_y$y ON gdelt.events (Year);"
   echo "CREATE INDEX idx_events_monthyear_y$y ON gdelt.events (MonthYear);"
   echo "CREATE INDEX idx_events_a1country_y$y ON gdelt.events (Actor1CountryCode);"
   echo "CREATE INDEX idx_events_a2country_y$y ON gdelt.events (Actor2CountryCode);"
done
```

Esta lista no es exhaustiva para los índices necesarios.

5 Análisis de los Datos

Se hizo un análisis simple de los datos cargados para resolver las especificaciones del proyecto 2.

5.1 UFO

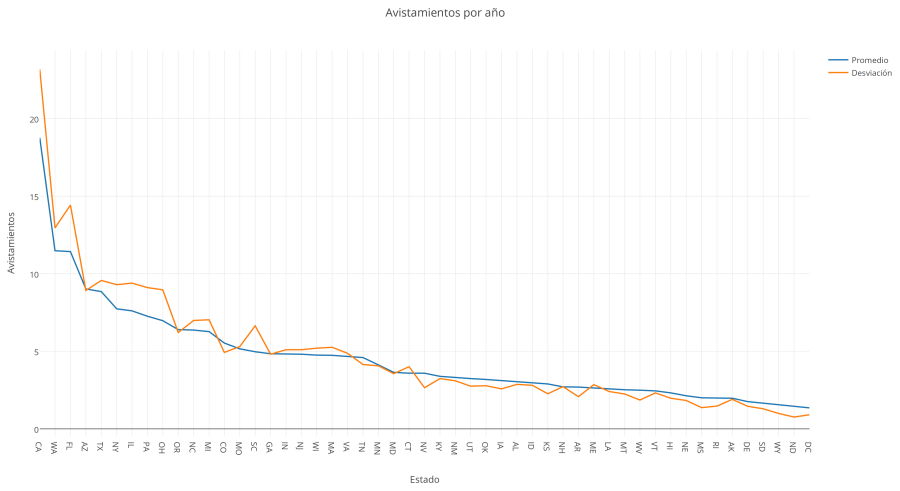
5.1.1 Primer avistamiento por estado

El comando en Posgres

```
select MIN(date_time), state_str  from sightings group by
↪  state_str;
```

No.	state	date	No.	state	date
1	AK	05/07/69	26	MT	15/01/
2	AL	01/07/69	27	NC	27/07/69
3	AR	21/04/69	28	ND	20/07/70
4	AZ	01/07/69	29	NE	01/03/69
5	CA	05/04/69	30	NH	15/10/69
6	CO	01/06/69	31	NJ	01/03/69
7	CT	01/05/69	32	NM	05/12/70
8	DE	15/10/69	33	NV	01/01/69
9	FL	04/01/69	34	NY	01/06/69
10	GA	01/04/69	35	OH	01/05/69
11	HI	15/04/69	36	OK	15/03/69
12	IA	10/07/69	37	OR	10/06/69
13	ID	24/07/73	38	PA	01/01/69
14	IL	01/03/69	39	RI	15/06/70
15	IN	15/02/69	40	SC	04/04/69
16	KS	15/08/69	41	SD	15/07/72
17	KY	01/06/69	42	TN	13/10/69
18	LA	01/01/69	43	TX	15/05/69
19	MA	30/06/69	44	UT	10/07/69
20	MD	01/05/69	45	VA	01/06/69
21	ME	27/08/69	46	VT	15/08/69
22	MI	01/06/69	47	WA	15/03/69
23	MN	01/07/69	48	WI	10/06/71
24	MO	15/05/69	49	WV	01/05/71
25	MS	01/06/69	50	WY	20/06/70

5.1.2 Promedio y desviación estándar de avistamientos anuales por estado



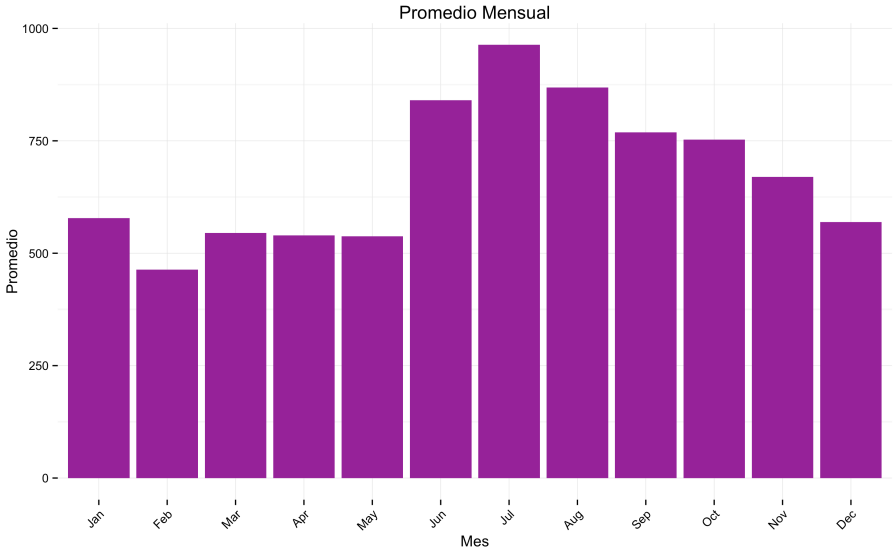
5.1.3 Primeros 15 avistamientos por figura

```
1 select MIN(date_time), shape from sightings group by shape;
```

shape	date
Flash	15/07/69
Fireball	11/07/69
Changing	30/06/69
Cone	30/06/69
Egg	30/06/69
Circle	01/06/69
Formation	01/06/69
Cylinder	21/04/69
Cross	15/03/69
Cigar	12/02/69
Disk	20/01/69
Light	15/01/69
Rectangle	15/01/69
Sphere	15/01/69
Oval	04/01/69

5.1.4 Promedio mensual de avistamientos

```
select month, avg(difference) from (select line_number, date_time,
↪  date_time - lag(date_time) over (order by date_time) as
↪  difference, date_part('month', date_time) as month from
↪  sightings) t group by month;
```



5.1.5 Tiempo promedio entre avistamientos por mes

El mes en el que tardan más los avistamientos es en enero mientras que agosto es el menor

```
select year, avg(difference) from (select line_number, date_time,
↪  date_time - lag(date_time) over (order by date_time) as
↪  difference, date_part('year', date_time) as year from
↪  sightings) t group by year order by year;
```

Month	Day	Hour	Minute	Seconds
01	3	21	02	49
02		10	33	06
03		08	32	57
04		13	53	32
05		08	52	08
06		14	06	30
07		07	37	40
08		05	09	20
09		06	55	31
10		07	54	42
11		05	43	42
12		15	24	25

5.1.6 Tiempo promedio entre avistamientos por estado

```
1 select state_str, avg(difference) from (select line_number,
  ↪  date_time, date_time - lag(date_time) over (order by
  ↪  date_time) as difference,   state_str from sightings) t  group
  ↪  by state_str ;
```

Los estados con menos de 5 horas entre avistamientos

Id	State
CA	California
NE	Nebraska
NH	New Hampshire
SC	South Carolina

Los estados con mayor tiempo promedio entre avistamientos

Id	State
ND	North Dakota
PA	Pennsylvania
AL	Alabama
FL	Florida

5.1.7 Cuál es el estado con mayor varianza

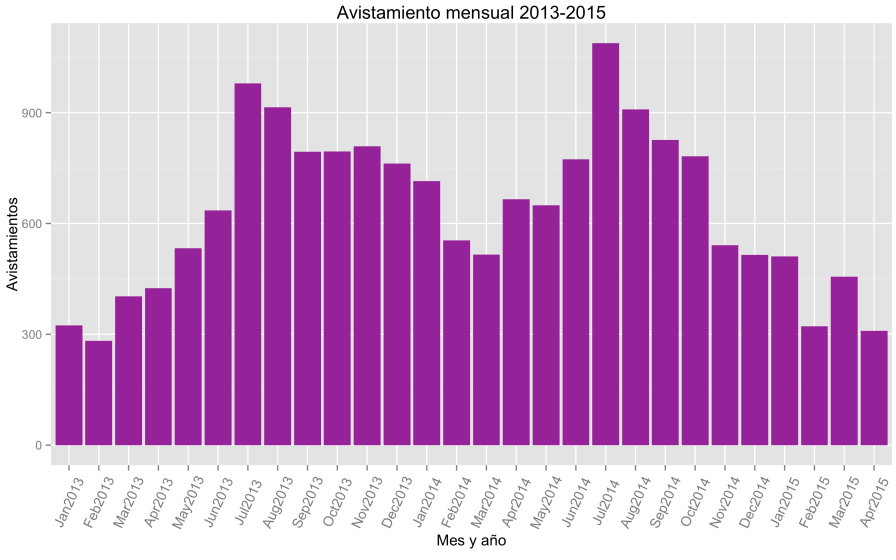
```
select state_str, variance(count) as variance from (select
↪  state_str, date_part('month', date_time) as month,
↪  date_part('year', date_time) as year, count(*) as count  from
↪  sightings group by state_str, month, year order by state_str,
↪  month, year) t group by state_str order by variance DESC;
```

Var	State
Min	Pennsylvania
Max	California

5.1.8 Existen olas temporales

```
select month, year, count(*) from (select line_number, date_time,
↪  year, month, extract(epoch from difference)/3600 as hour_diff
↪  from (select line_number, date_time, date_time -
↪  lag(date_time) over (order by date_time) as difference,
↪  date_part('year', date_time) as year, date_part('month',
↪  date_time) as month from sightings) t) t1 where 24< hour_diff
↪  and hour_diff <= 48 group by month, year order by year, month;
```

Para esta sección lo truncamos a los últimos dos años para ver si existía algún patrón en los avistamientos. Como podemos ver en la gráfica en Julio hay más avistamientos, tal vez debido a la celebración de la independencia de Estados Unidos. El estado que más avistamientos tiene es California



5.1.9 Existen olas espacio- temporales

```
select state_str, month, year, count(*) as count from (select
↪ line_number, date_time, state_str, year, month, extract(epoch
↪ from difference)/3600 as hour_diff from (select line_number,
↪ state_str, date_time, date_time - lag(date_time) over (order
↪ by date_time) as difference, date_part('year', date_time) as
↪ year, date_part('month', date_time) as month from sightings)
↪ t) t1 where 24< hour_diff and hour_diff <= 48 group by
↪ state_str, month, year order by state_str, count DESC;
```

5.1.10 Narrativas parecidas

```
select date_time, city_str, state_str, similarity(summary,
↪ lag(summary) over (order by date_time, state_str)) as simil
↪ from sightings order by state_str, date_time;
```

6 Conclusiones

Este proyecto fue muy ilustrativo para destacar las ventajas que tiene Postgres para el manejo de grandes bases de datos:

- Particionamiento de tablas
- Índices especializados e índices parciales
- Librerías para búsquedas geo localizadas
- Librerías para búsquedas en texto difuso

Además fue un caso práctico para los procesos de automatización de adquisición, limpieza y carga de datos de distintas fuentes.