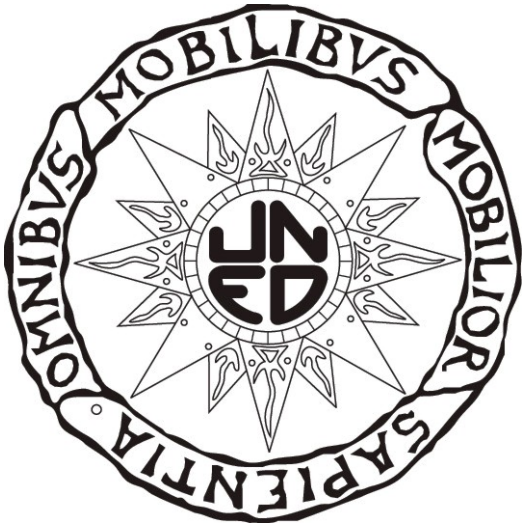


SISTEMAS DE BASES DE DATOS

Introducción al manejo de datos masivos con Hadoop

INDICE

1.- INTRODUCCIÓN

El trabajo solicitado nos establece varios marcos de trabajo, el tratamiento de scripts en python y mrjob, el lenguaje de consultas SQL, la comprensión de computación paralela distribuida y manejo de los frameworks propuestos como herramientas Hue, Hive,… que vienen configuradas en la máquina virtual de Cloudera, para el tratamiento de datos masivos con Hadoop.

Toda esa información, viene bastante bien introducida en el enunciado de la práctica, que junto con los videos elaborados y facilitados en el alf de la asignatura, complementa sin lugar a dudas lo expuesto ella.

El escenario elegido para la elaboración del trabajo no ha sido Cloudera, dado que no disponía de una máquina que moviera con soltura, pero pude configurando un i3 con 4gb de ram, arrancar la maquina virtual, asignándole 2 núcleos y 3gb de RAM y un arranque desacoplado. Pero aún así aunque no se movía con soltura pude hacer unos ejercicios de prueba siguiendo los vídeos y la teoría del enunciado, usando los scripts en python de ejemplo para hacer un preprocesamiento de los datos, y subir los ficheros en el cluster de Cloudera.

Las ordenes más relevantes fueron:

hadoop fs -mkdir nombreCarpeta

para crear un carpeta en el cluster.

hadoop fs -ls [carpeta]

para listar el contenido de las carpeta

hadoop fs -put origen destino

para subir un fichero al sistema de archivos HDFS (al cluster vamos)

hadoop fs -get origen destino

para bajar un fichero del sistema de archivos

Como podemos apreciar los comandos son similares a los utilizados en lineas de comandos linux y/o sistemas adb del sistema android. Una documentación más desarrollada la encontramos en la página de soporte de Apache Hadoop y HDFS.

Por otro lado no podemos desviar la atención de las operaciones MapReduce utilizando los scripts en python, distinguiendo las 3 fases Map – Shuffle & Sort – Reducer. Los cual podemos traducir en el mapper – reducer que podremos usar en ficheros de script python que deben subir al cluster y que serán repartidos a los distintos bloques en los que están repartidos los datos del fichero que contienen la información a procesar, viéndose así la computación paralela distribuida, usando el jar de hadoop:

jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar

al cual debemos pasar los fichero mapper.py y reducer.py, origen de datos y destino de salida de la tarea MapReduce tal y como podemos ver en la página 17 del enunciado de la practica.

Cuando hadoop termina la tarea MapReduce genera un fichero SUCCESS de confirmación y un fichero part-00000 en el destino indicado, el cual podríamos recuperar desde Hue si usamos la web o bien desde la consola con el -get de haddop. Cada nodo que contiene cada bloque necesita los scripts para procesar dicha información.

Por otro lado y como requisito en los ejercicios 1, 2 y 3. podemos ver como esa misma actividad se puede hacer con un fichero compuesto por las secciones mapper-combiner-reducer importando la librería con mrjob previamente instalado con pip de python. Con el uso de mrjob

podemos lanzar el script contra un cluster hadoop o bien de forma local sin usar hadoop, que será la forma que usaremos en el desarrollo de las tareas, de forma local sin hadoop.

2.- ESCENARIO DEL TRABAJO Y HANDICAPS

Como se ha comentado en la introducción, en la máquina i3 con 4gb y windows 7 64bits, no funcionaba fluida la maquina virtual de Cloudera. Se pudieron lanzar algunos ejemplos de los videos del alf, pero el rendimiento no era suficiente para abordar el trabajo. Entonces se descargó la máquina virtual propuesta para realizar la práctica con ubuntu y postgresql. Pero los entornos virtuales no son de mi agrado y como habitualmente trabajo con linux en el portátil, pues instale para la práctica la distribución de Linux 16.04.03 para poder trabajar cómodamente. El entorno lo prepare con los siguientes paquetes:

- pip-python
- mrjob
- postgresql
- apache
- php
- phppgadmin

Antes de ello también intente instalar con el manager de Cloudera el entorno de Hadoop y poder usar Hue y Hive, pero comenzó a dar errores difíciles de debugear a la hora de configurar los clusters y desistí de ello, lo cual sin duda es interesante y que intentare en otra ocasión por la razón que expondré en las conclusiones.

Paso a repasar los pasos que seguí para que la memoria sea autocontenida en la medida de lo posible:

```
$ sudo apt-get install python-pip
$ sudo pip instal mrjob
$ sudo apt-get install  postgresql postgresqp-contrib
```
La instalación de postgresql crea al usuario postgres y para cambiar la cuenta al server usaríamos

```
$ sudo -i -u postgres
```
o bien
```
$ sudo -u postgres psql
```

Para abrir la consola de postgresql usamos
```
$ psql
```
Aunque realmente el acceso lo realizaremos con:
```
$ psql -U postgres -W -h localhost nombreBaseDatos
```

Siendo postgres el usuario, localhost la máquina a la que acceder y nombreBaseDatos la base de datos a la que “atacar”.

Más abajo se realizarán algunas consideraciones más para el manejo con POSTGRESQL desde la terminal, por ejemplo algunos comando, aclaración de tipos y opciones para colgar los datos de los ficheros en las tablas que vamos creando.

Con lo anterior tendríamos suficiente configurado el entorno. Pero Cloudera nos ofrece un interfaz web para el manejo de las bases de datos muy atractivo, HIVE, y que otros motores como mysql también existen administradores web como phpmyadmin que nos da un entorno amigable de manejo de esas bases de datos. Para SQLite3 (usado entre, otros sistemas, en Android, ya que la mayoría de las apps, por ejemplo de mensajería, hacen uso) tenemos por ejemplo un plugin para Mozilla Firefox muy interesante. Para postgresql podemos encontrar phppgadmin, que

requiere un servidor web instalado (en nuestro caso apache) y php. El cual preparamos con para un uso básico, sin seguridad ni carpetas ni nada, tal y como nos la el configurador.

```
$ sudo apt-get install apache
$ sudo apt-get install php7.0 libapache2-mod-php7.0
$ sudo apt-get install phppgadmin
```

Con esta configuración básica solo permite trabajar en localhost, que si quisiéramos sacar tendríamos que comentar Require local del /etc/apache2/conf-avalable/phppgadmin.conf y añadir a localhost la ip actual, por defecto el puerto es el 5432.

Se añade un resumen de comandos POSTGRESQL entre los ficheros de la resolución de la practica, que no incluimos por la limitación de espacio exigida para la memoria. Pero si vamos a mencionar algunos para iniciarnos en el manejo de la consola.

Señalar que la contraseña del usuario postgres se cambia con:

```
$ sudo su postgres
$ psql
=# alter user postgres with password ‘nuevaPass’
```

Los siguientes comando se pueden ejecutar desde el interprete de postgresql

=# \q para salir
=# \d muestra las tablas
=# \l muestra las bases de datos si no hay ninguna seleccionada.
=# \h muestra la ayuda
=# \h comando muestra la ayuda mas extensa del comando del que pedimos ayuda con la explicación de los parámetros

Para crear tablas usaremos la sintaxis similar de SQL, pero prestando atención a la declaración de tipos de los campos, nombraremos los mas relevantes para nuestro objetivo:

- Los strings: varchar(x), text y char(x), siendo los dos primeros de tamaño variable y el último de tamaño fijo. Elegimos por comodidad TEXT.
- Los enteros: int2, int4 y int8. Correspondientes a smallint, int y bigint conocidos de otros lenguajes y escenarios. Si lo necesitásemos usaríamos INT4.
- Los flotantes: real y float8, que corresponderían a float y double de otros lenguajes. Usaremos REAL.

Mencionaremos en este apartado del escenario el uso de COPY el cual veremos en la tarea 4 para poder cargar el fichero de datos preprocesado a las tablas. El delimitador de campos al ser un tabulador ‘\t’ hay que escaparlo con la letra E, el resto de delimitadores no son necesarios en la gran mayoría escaparlos. La sentencia copy para subir y recoger datos desde/hacia los ficheros vendrán en las formas respectivas con FROM y TO, quedaria como:

```
COPY enLaTabla FROM ‘rutaDelFichero’ WITH (DELIMITER E’\t’)
```

Otro de los handicaps del escenario, aparte de la configuración de éste, ha sido lidiar con la consola de postgresql, ya que no la había usado anteriormente, no así con mysql, con la que estaba mas familiarizado. Pero sin lugar a dudas si seguimos los videos o la documentación del enunciado, vemos que los tipos no coinciden, lo cual dificulta la elaboración del ejercicio 4, ya que primordialmente requeriría un conocimiento del dominio de los datos que vamos a procesar, y realmente, eso de bioinformática, como que no me dice nada actualmente, después de varios experimentos, intente usar los tipos más generales que pude con los cuales pudiese realizar operaciones matemáticas, REAL, y lo que no pues a TEXT.

Añadimos otros 2 handicaps más añadiendo el uso de python y el de mrjob, y como no el concepto de generadores y **yield** algo que se asemeja a la evaluación perezosa, la construcción de listas y uso de rangos de haskell estudiados en la asignatura de Teoría de Lenguajes de Programación, en el apartado de programación funcional. Pero que una vez comprendido su funcionamiento, parece útil para el aprovechamiento de recursos, ya que generamos lo datos que vamos necesitando, o por lo menos esa es la impresión.

No podemos pasar a exponer las tareas o ejercicios sin volver a mencionar, que queda comprendido que los scripts que vamos usar de mrjob, nos permiten trabajar de forma similar a como lo haríamos si usásemos Hadoop, con el beneficio de no tener que instalarlo en este entorno y que esos scripts podríamos lanzarlos directamente contra hadoop con -r hadoop o localmente con -r local.

En este ejercicio vamos a trabajar con datos del Gen Expression Omnibus (GEO): https://www.ncbi.nlm.nih.gov/geo/info/geo_paccess.html#Programs

En concreto, utilizaremos el fichero GSD1001_full.soft.txt (que se puede descargar del curso virtual) que presenta los datos de una serie de experimentos en Página 26 INTRODUCCIÓN AL MANEJO DE DATOS MASIVOS CON HADOOP bioinformática. El contenido del fichero es una serie de filas con las siguientes columnas separadas por tabuladores:
•ID_REF - IDENTIFIER - GSM19023 - GSM19024 - GSM19025 - GSM19026 - Gene title - Gene symbol - Gene ID - UniGene title - UniGene symbol - UniGene ID - Nucleotide Title – GI - GenBank Accession - Platform_CLONEID - Platform_ORF - Platform_SPOTID - Chromosome location - Chromosome annotation - GO:Function - GO:Process - GO:Component - GO:Function ID - GO:Process ID - GO:Component ID

Sobre estos datos es necesario realizar algunas operaciones bien utilizando MapReduce directamente o bien utilizando consultas Hive. Las tareas a realizar son las siguientes:

3.- TAREA 1

Partiendo del fichero GSD1001_full.soft.txt, preprocésalo para eliminar las columnas que no sean de nuestro interés, quedándonos con las 13 primeras columnas del fichero original. Esto se realiza con un trabajo MapReduce. Las columnas que nos interesan son las siguientes: "idref", "ident", "gsm19023", "gsd19024", "gsd19025", "gsd19026", "genetitle", "genesymbol", "geneID", "uniGenetitle", "uniGenesymbol", "uniGeneID", "NucleotideTitle". Además, tenemos que incluir tres nuevas columnas, llamadas “max”, “min” y “avg” que contendrán datos numéricos resultado de calcular el valor máximo, mínimo y medio de las columnas " gsm19023", "gsd19024", "gsd19025" y "gsd19026" para cada fila, respectivamente. De esta forma, el resultado de este punto es un fichero que llamaremos DataClean.txt que tendrá 16 columnas. Para implementar este punto es necesario utilizar la librería mrjob.

Los ficheros del Dataset, **GDS1001_full.soft.txt** , el fichero del código fuente, **tarea1.py**, las columnas y el **DataClean.txt** generado en esta tarea están en la carpeta **tarea1**, junto con algunas capturas y el de ordenes.

La generación del fichero DataClean.txt con las 16 columnas, ha sido generado desde la una terminal abierta en esa carpeta con la orden:

```
$ python tarea1.py GDS1001_full.soft.txt -r local > DataClean.txt
```

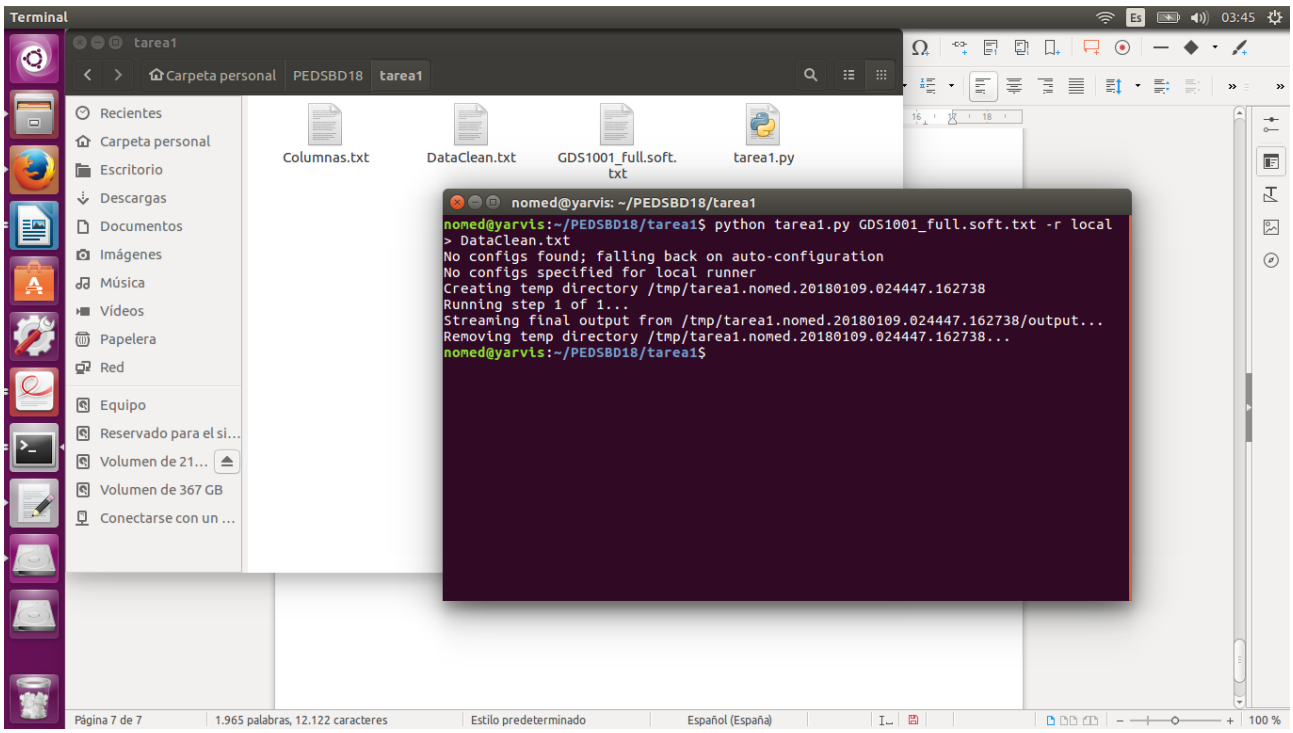


Figura 1.- Orden de procesamiento de la Tarea1

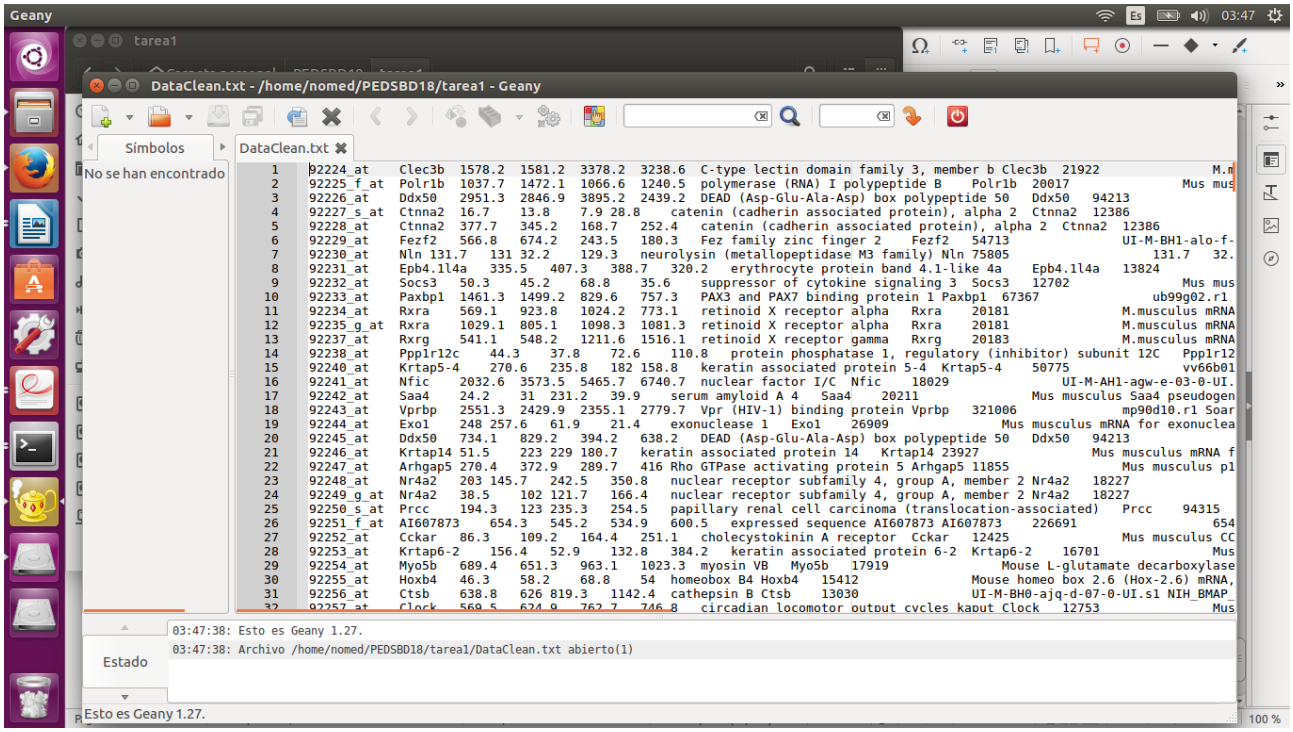
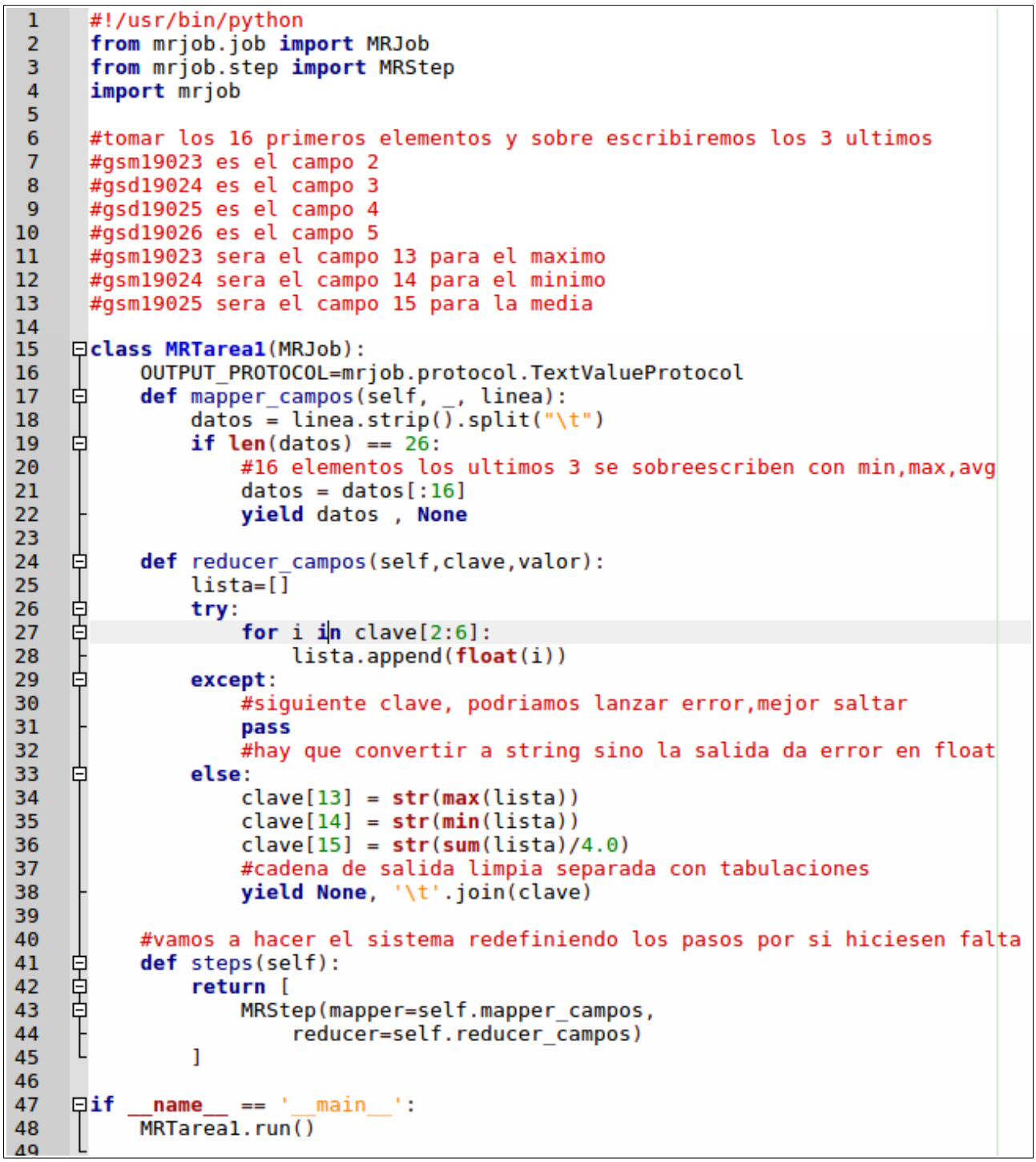


Figura 2.- Resultado del fichero DataClean.txt



```python
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
import mrjob

#tomar los 16 primeros elementos y sobre escribiremos los 3 ultimos
#gsm19023 es el campo 2
#gsd19024 es el campo 3
#gsd19025 es el campo 4
#gsd19026 es el campo 5
#gsm19023 sera el campo 13 para el maximo
#gsm19024 sera el campo 14 para el minimo
#gsm19025 sera el campo 15 para la media

class MRTarea1(MRJob):
    OUTPUT_PROTOCOL=mrjob.protocol.TextValueProtocol
    def mapper_campos(self, _, linea):
        datos = linea.strip().split("\t")
        if len(datos) == 26:
            #16 elementos los ultimos 3 se sobreescriben con min,max,avg
            datos = datos[:16]
            yield datos , None

    def reducer_campos(self,clave,valor):
        lista=[]
        try:
            for i in clave[2:6]:
                lista.append(float(i))
        except:
            #siguiente clave, podriamos lanzar error,mejor saltar
            pass
            #hay que convertir a string sino la salida da error en float
        else:
            clave[13] = str(max(lista))
            clave[14] = str(min(lista))
            clave[15] = str(sum(lista)/4.0)
            #cadena de salida limpia separada con tabulaciones
            yield None, '\t'.join(clave)

    #vamos a hacer el sistema redefiniendo los pasos por si hiciesen falta
    def steps(self):
        return [
            MRStep(mapper=self.mapper_campos,
                reducer=self.reducer_campos)
        ]

if __name__ == '__main__':
    MRTarea1.run()
```

Figura 3.- Script python de la Tarea1

Como vemos el código no hemos usado los nombres de los campos como variables, puesto que no necesitamos mostrarlos por pantalla. Ahora bien en los comentarios al inicio si que hay una descripción de los que vamos a usar. Importamos mrjob. Creamos la clase que vamos manejar y añadimos la directiva de formato de salida Texto sin formato en OUTPUT_PROTOCOL. Nos definimos las funciones del MapReduce, que hemos llamado mapper_campos y reducer_campos y en la definición de steps, indicamos cual será el mapper y el reducer. Se ha hecho así simplemente para resaltar el formato que se podría usar si quisiémos o la tarea lo exigiese para procesar uno o mas pasos, esto viene documentado en el manual de mrjob. Perfectamente podríamos haber llamado mapper y reducer las funciones y olvidarnos en este caso de la definición de steps. Vemos como tomamos una linea, comprobamos que hay 26

campos con tabulación separados y si no es así obviamos esa línea. Es linea la spliteamos con el carácter de tabulación y después tomamos 16 campos, los 3 últimos los vamos a sobrescribir. Pero así tendremos 16 campos (13 + max+min+avg), numerados de 0 a 15. Los cuales devolvemos como clave y en forma de generador gracias a yield. El cual funciona como un “return” pero que entra en bucle (ahorrándonos escribir código, gracias a mrjob) y va devolviendo tuplas conforme las vamos necesitando, con el correspondiente ahorro de recursos.

La entrada al reducer serán los elementos en forma de lista que salen del mapper, tomamos los campos 2,3,4,5 e intentamos prepararlos como float para poder operarlos matemáticamente, y si todo va bien, calculamos el max, min y la media y la almacenamos en forma de cadena, para su salida con yield los cuales se guardaran en el fichero DataClean según la orden de lanzamiento del script. Se ha usado join() para dar una salida limpia y tabulada, ofreciendo el resultado que vemos en la Figura 2. Cumpliendo la tarea solicitada.

4.- TAREA 2

Partiendo del fichero DataClean.txt creado en el punto anterior, implementa un trabajo MapReduce que devuelva el mayor “avg” de aquellas filas cuyo valor de "gsm19023" esté entre 100 y 1000. Para implementar este punto es necesario utilizar la librería mrjob.

Los ficheros **DataClean.txt**, resultado de la tarea anterior, el fichero de **ordenes** para la consola, el fichero del código para el script **tarea2.py** y el fichero de salida **output**, están en la carpeta **tarea2**.

La generación de la salida en el fichero output ha sido generada desde la una terminal abierta en esa carpeta con la orden:

$ python tarea2.py DataClean.txt -r local > output

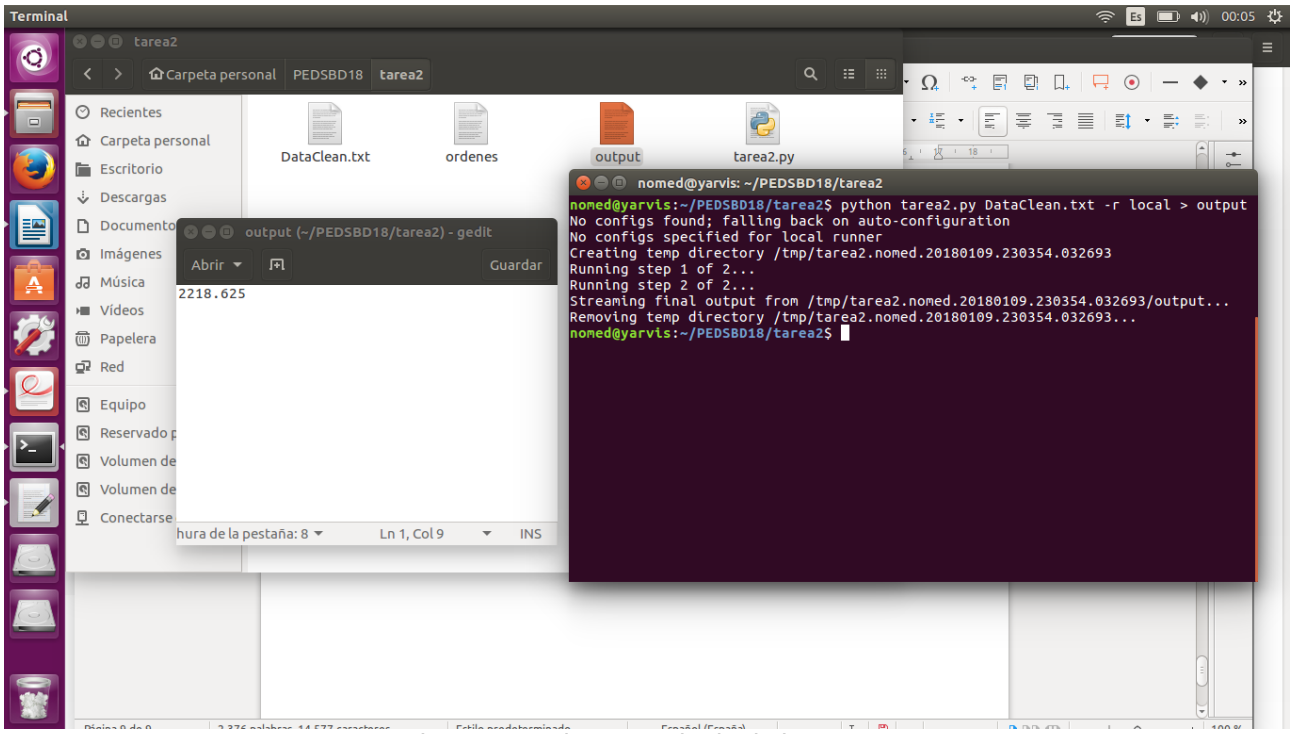


Figura 4.- Orden y resultado de la Tarea2

```python
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
import mrjob

#gsm19023 es el campo numero 2
#la media es el campo numero 15
class MRTarea2(MRJob):
    OUTPUT_PROTOCOL=mrjob.protocol.TextValueProtocol
    def mapper_campos(self, _, linea):
        datos = linea.strip().split("\t")
        #no haria falta comprobar pero por si acaso
        if len(datos) == 16:
            try:
                dato = float(datos[2])
            except:
                pass
            else:
                if 100 <= dato <= 1000:
#                    yield datos,None
                    yield datos,None

    def reducer_campos(self,clave,valor):
        yield None,float(clave[15])

    #recibe una lista de pares, devuelve el primer valor de la tupla max
    def reducer_max_media(self,_,valor):
        yield None, str(max(valor))

    #dos pasos, en el segundo paso es una lista de pares
    def steps(self):
        return [
            MRStep(mapper=self.mapper_campos,
                reducer=self.reducer_campos),
            MRStep(reducer=self.reducer_max_media)
        ]

```

Figura 5.- Script python de la Tarea2

Como hicimos en la tarea anterior indicamos los campos a usar en los comentarios en el inicio del código y hemos declarado una salida de texto en el protocolo, esta vez si podemos ver hemos trabajado en dos pasos la realización del trabajo usando las dos invocaciones de MRStep indicando cuales serán las funciones mapper y reduce a usar exigidas para mrjob. El mapper realiza un split con la tabulación como delimitador, devolviendo una lista con los valores de los campos, a cada línea, eligiendo solamente los que verifican la condición de ser mayor que 100 y menos que 1000, pasando los valores como generador a la siguiente fase de reducer_campos. Se trata una excepción al intentar convertir el tercer campo (índice numero 2) en formato float para poder comparar NUMÉRICAMENTE, ya que se pueden comparar cadenas lo que daría un resultado 999.9 erróneo en forma de cadena, no de número, al ejecutar la función max sobre el generador de la lista de valores, que después convertimos a string, dándonos así el valor requerido en el enunciado.

Como vemos tenemos dos funciones reducer, correspondientes a 2 pasos (steps) el primero ira generando la lista requerida por el reducer del segundo paso, que podrá hacer un uso correcto de max, para obtener el valor máximo del conjunto de la lista generada.

5.- TAREA 3

Partiendo del fichero DataClean.txt, implementa un trabajo MapReduce que devuelva la media del campo "gsd19025" teniendo en cuenta todas las filas de dicho fichero. Para implementar este punto es necesario utilizar la librería mrjob.

Los ficheros DataClean.txt, resultado de la tarea1 y también usado en la tarea2, el fichero de ordenes para la consola, el fichero del código para el script tarea3.py y el fichero de salida output, están en la carpeta tarea3.

La generación de la salida en el fichero output ha sido generada desde la una terminal abierta en esa carpeta con la orden:

```
$ python tarea3.py DataClean.txt -r local > output
```

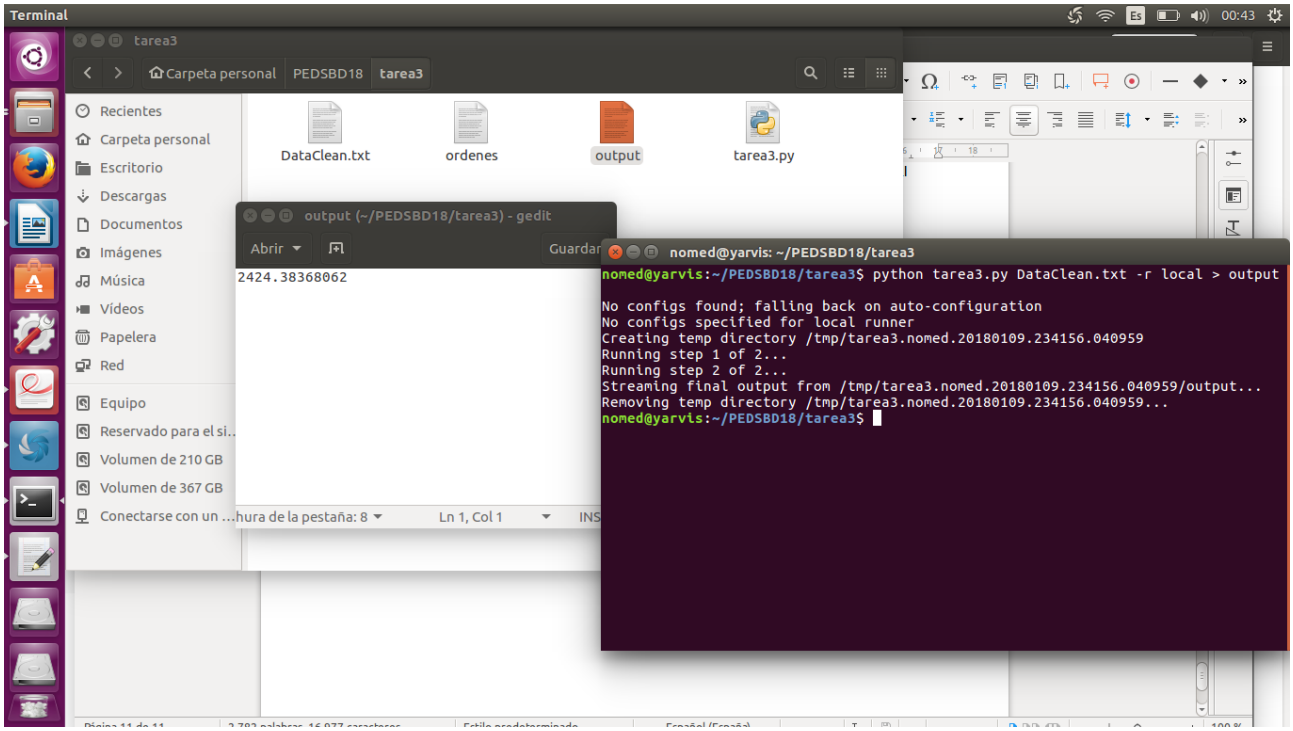


Figura 6.- Orden y resultado de la Tarea3

De la misma que en la tarea 2 se inicia el código del script, utilizando generadores con yield en el mapper, que pasan a la siguiente fase, el reducer. En el segundo paso de proceso, el reducer va a recibir una lista de generadores para los elementos del anterior paso, el cual va contando los elementos en cada iteración del for que va cogiendo cada elemento para sumarlo al total y poder con el contador obtener la media exigida.

```python
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
import mrjob

#gsd19025 es el campo numero 4
class MRTarea3(MRJob):
    OUTPUT_PROTOCOL=mrjob.protocol.TextValueProtocol
    def mapper_campos(self, _, linea):
        datos = linea.strip().split("\t")
        #no haria falta comprobar pero por si acaso
        if len(datos) == 16:
            try:
                gsd19025 = float(datos[4])
            except:
                pass
            else:
                yield datos,None

    def reducer_campos(self,clave,valor):
        yield None,float(clave[4])

    #recibe una lista , los contamos y asi sabemos el numero de elemento
    def reducer_media(self,_,valores):
        suma=0
        cont=0
        for dato in valores:
            suma = suma + dato
            cont = cont + 1
        yield None, str(suma/cont)

    #dos pasos, en el segundo paso es una lista de pares
    def steps(self):
        return [
            MRStep(mapper=self.mapper_campos,
#                combiner=self.combiner_campos,
                reducer=self.reducer_campos),
            MRStep(reducer=self.reducer_media)
        ]

if __name__ == '__main__':
    MRTarea3.run()
```

Figura 7.- Script python de la Tarea3

6.- TAREA 4

Partiendo del fichero DataClean.txt, crea una tabla Hive interna y otra externa. El resultado de este punto son dos tablas que llamaremos datacleantableinternal y datacleantableexternal.

Como el entorno utilizado no nos permite utilizar Hive, lanzaremos los comandos directamente desde la consola de comando de postgresql. Usaremos también una interfaz gráfica web en la dirección http://localhost/phppgadmin para ver mejor organizada la información, aunque todos los comandos usando el botón SQL de la interfaz web podemos usarla para lanzar los comandos, tal y como puede verse en el video proporcionado de HIVE.

Lo primero que haremos es ver que si lanzamos desde la consola el comando psql desde la terminal para abrir el postgresql, dará un error porque nuestro usuario no esta permitido para usarlo. Deberíamos darle permisos o meterlo en el grupo de usuarios permitidos. No compliquemos la cosa y lo que haremos sera entrar como postgres con permisos de

superususario, y mostrar algo importante que es cambiar la contraseña de **postgres** y ponerle **sbd2018user**.

Después crearemos la base de datos **ped2018**. A continuación creamos la tabla dataclean con los 16 campos y los tipos que he creído convenientes, dado el poco conocimiento del dominio de bioinformática que poseemos, pero si suficientes para adaptar el trabajo y poderlo realizar satisfactoriamente. Como expuse en la sección del escenario (punto 2) en la cual se introduce unas pequeñas aclaraciones a tener en cuenta.

```
$ sudo su postgres
$ psql
=# alter user postgres with password ‘sbd2018user’
```

La creación de la base de datos será:

```
=# create database ped2018
```

Después de crear la tabla usaremos el comando COPY para cargar los datos en la tabla.

```
COPY dataclean FROM '/home/nomed/PEDSBD18/tarea4/DataClean.txt' WITH (DELIMITER E'\t');
```

donde con dataclean es la base de datos la ruta del fichero de datos DataClean.txt e indicamos el delimitador escapado para la tabulación, que representamos con la E para indicar que es un carácter de escape y el carácter de tabulación ‘\t’. Se pude ver gráficamente todo lo expuesto en la figura 8 y 9.

Todos los ficheros de este ejercicio están en la carpeta Tarea4, las ordenes y los ficheros que corresponde, junto con las imágenes.

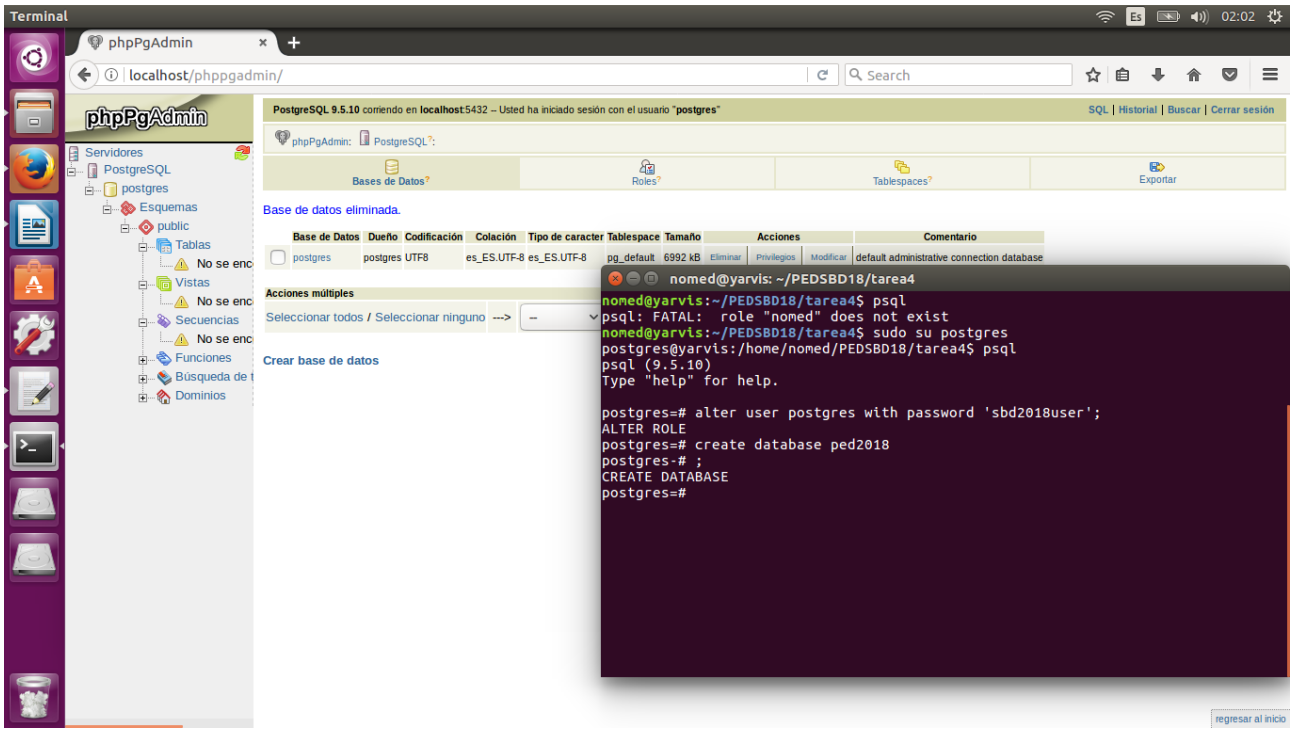


Figura 8.- Acceso a postgresql, cambio de pass y creación de la base de datos ped2018

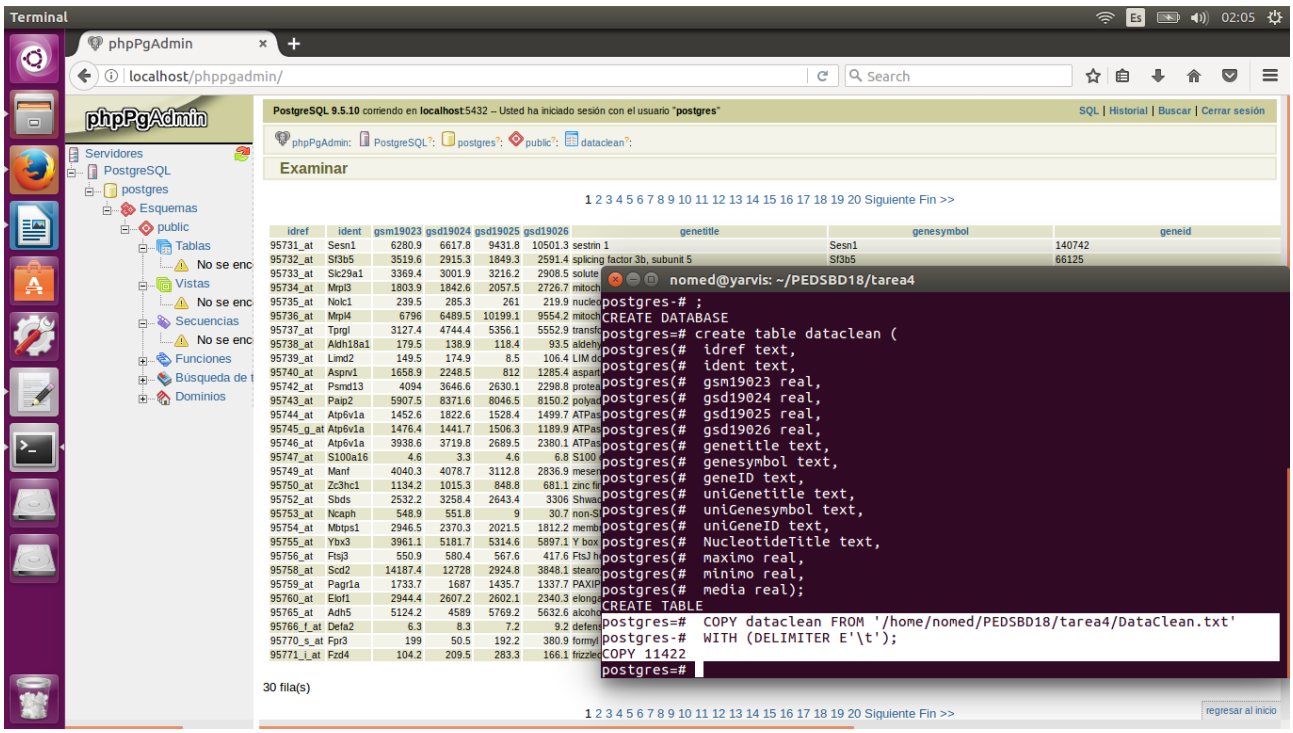


Figura 9.- Creación de la tabla y subida de los datos desde el DataClean.txt

7.- TAREA 5

Para volver a entrar a postgres a la base de datos ped2018 y poder trabajar con ella lo haremos con la siguiente orden y el pass bsd2018user:

```
$ psql -U postgres -W -h localhost ped2018
```

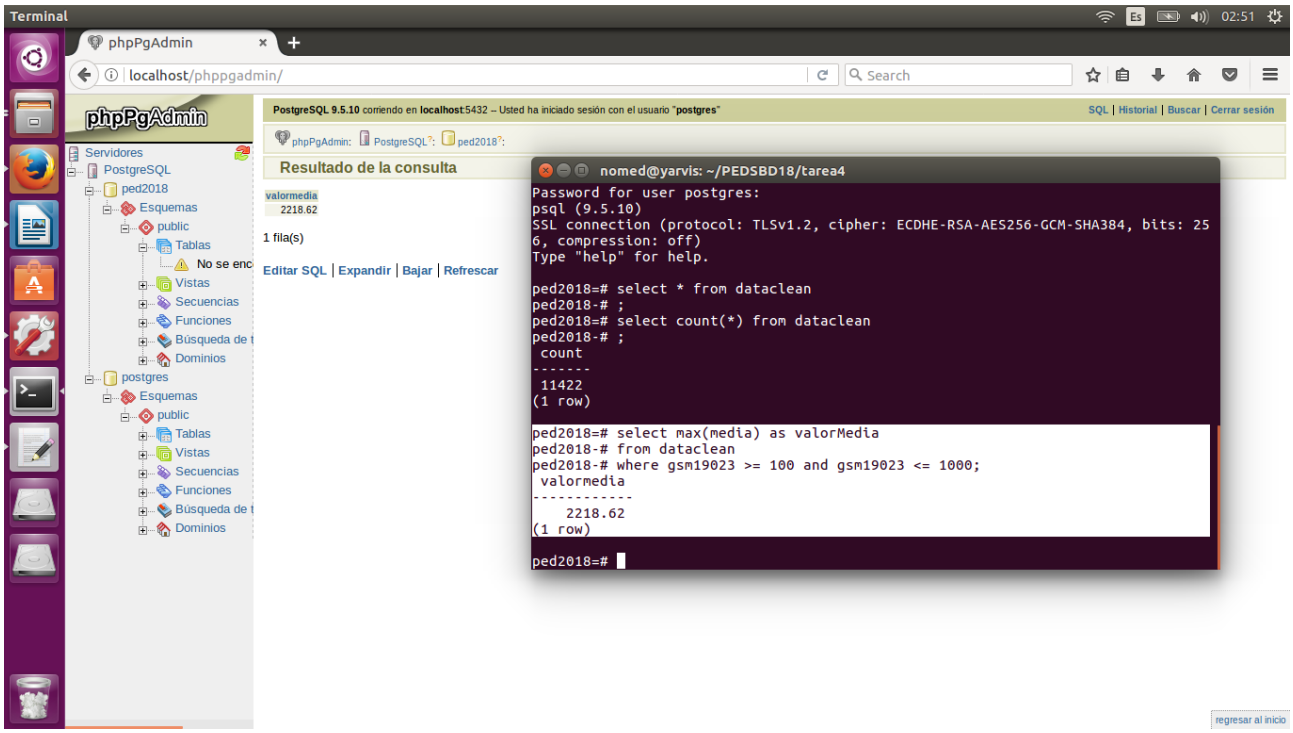


Figura 10.- Ejercicio a de la Tarea 5

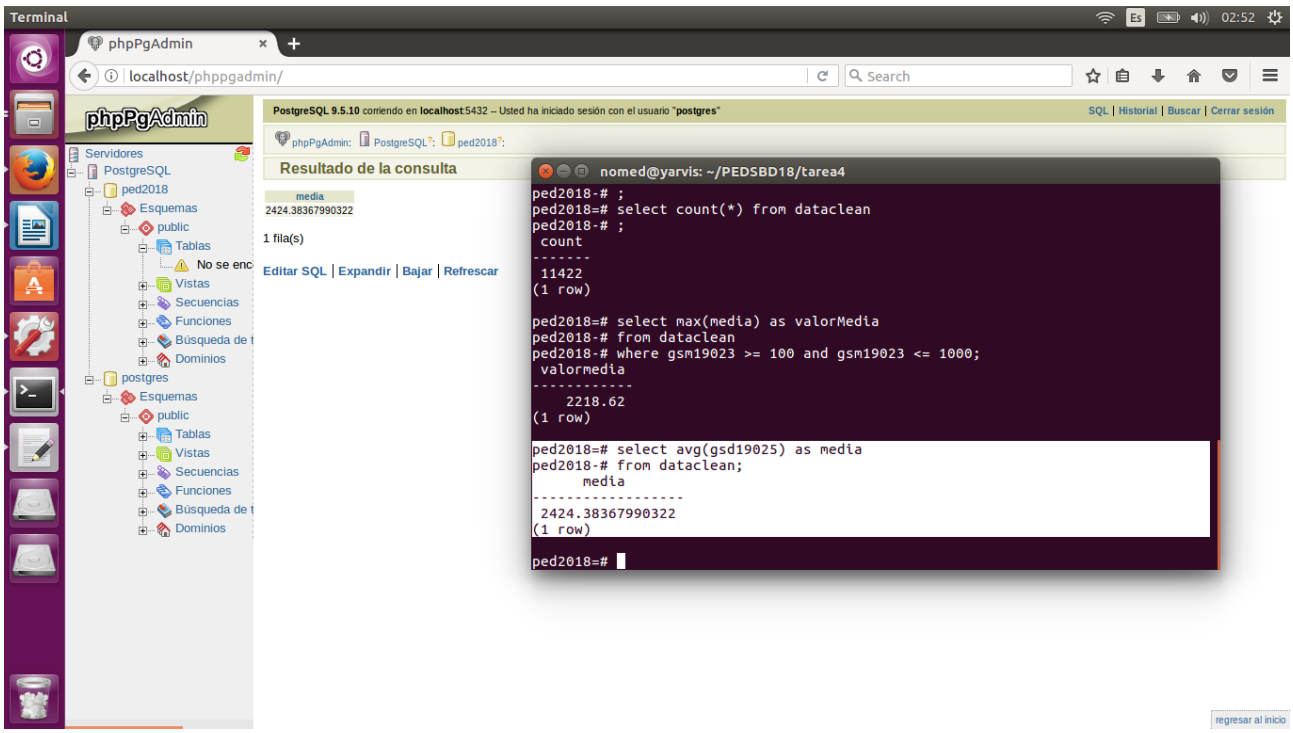


Figura 11.- Ejercicio b de la Tarea 5

Comandos postgresql para los ejercicios a y b de la tarea 5.

```
select max(media) as valorMedia
from dataclean
where gsm19023 >= 100 and gsm19023 <= 1000;

select avg(gsd19025) as media
from dataclean;
```

Y como se puede ver en la figuras 10 y 11, los resultados corresponden con los obtenidos en las tareas 2 y 3, en las figuras 4 y 6.

8.- CONCLUSIONES

El trabajo ha sido interesante gracias a los requisitos para elaborarlo, el cual ha exigido una introducción a python y a mrjob, ha sido el primer contacto con el lenguaje. El entorno usado con postgresql,no ha sido complicado por haber trabajado antes con mysql, de una forma similar, tan solo atención a las órdenes básicas y los tipos y la orden de subida de datos.

En cuanto a Cloudera, la explicación y videos aportados han sido satisfactorios y lo empleare en breve, en unos proyectos open source que estoy desarrollando para informática forense con SmartPhone basados en Android. Tomando las bases de datos SQLite3 del sistema, procesare y subiré a Cloudera, para poder aprovechar la funcionalidad del Mapa, como aparece en la página 24, figura 20 y poder establecer lugares con las cronologías de “sucesos” aprovechando las funcionalidades de posicionamiento de los smartphone, ideal.

* * *