Infraestructura para Big Data
Evaluación de prestaciones

Daniel Pérez Efremova

Enero 2022

Índice

Aclaraciones

Para realizar la práctica se ha utilizado la traza proporcionada en el curso. En la sección de ejercicios se presenta la idea detrás de cada consulta, el código ejecutado y una tabla que recoge las primeras filas del resultado.

Ejercicios

Ejercicio 1. Crear un script HIVE que obtenga la serie temporal en bits/s con granularidad 1 segundo para la traza de análisis. (0,75 puntos)

Para obtener la serie se agregan todos los paquetes por timestamp y se suma la longitud multiplicada por ocho para obtener el dato en bits.

Código 1: Consulta para obtener la serie

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SELECT ts, sum(len)*8 as bits
from pcaps
group by ts
order by ts;
```

En la tabla 1 se muestran unas líneas del resultado.

Tabla 1: Ejemplo del resultado de la consulta

timestamp	bits
1511781613	1216
1511781621	1280
1511781622	1616
1511781630	1648
1511781635	NULL
1511781643	1184
1511781651	2752
1511781654	1104

Se observa que en algunos segundos del timestamp se ha producido algún error

ya que hay valores NULL en la columna bits. Además, la serie no es continua ya que no estan todos los segundos en las filas.

Ejercicio 2. Crear un script HIVE que obtenga registros de flujos junto con el número de bytes y paquetes para la traza de análisis. Los registros de flujos han de contener: Dirección IP origen y destino, Puerto origen y destino, Protocolo (TCP o UDP), Número de bytes, Número de paquetes. (0,75 puntos)

Para obtener el registro de flujos se agregan las observaciones por origen, destino y protocolo para despues calcular la cantidad de bytes y paquetes.

Código 2: Consulta para obtener el registro de flujos

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SELECT src,
        dst,
        protocol,
        sum(len) AS bytes,
        count(*) AS packages
FROM pcaps
WHERE protocol IN ('UDP', 'TCP')
GROUP BY src, dst, protocol;
```

En la tabla 2 se observa un ejemplo del registro.

Tabla 2: Ejemplo del registro

src	dst	protocol	bytes	packages
8.254.173.126	192.168.158.128	TCP	63212058	22610
192.168.158.128	8.254.173.126	TCP	434872	18960
130.206.192.17	192.168.158.128	TCP	16135936	7766
192.168.158.128	192.168.158.2	UDP	254928	6988
192.168.158.2	192.168.158.128	UDP	840238	6940

Ejercicio 3. Crear un script HIVE que obtenga los 'Host' (a nivel HTTP) más populares (en número de peticiones) en la traza de análisis. (0.75 puntos)

En este ejercicio se precisa ampliar la tabla provista en el curso con el campo *header_host*. Una vez añadido, basta hacer un recuento de peticiones (*http_request*) agregando por los valores de host y ordenar descendentemente para obtener el ranking de hosts más populares.

Código 3: Consulta para obtener el ranking de hosts

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SET net.ripe.hadoop.pcap.io.reader.class=
net.ripe.hadoop.pcap.HttpPcapReader;
SELECT header-host, count(http-request) as count
        FROM http
        GROUP BY header-host
        ORDER BY rank DESC LIMIT 10;
```

En la tabla 3 se muestra un ejemplo del resultado.

Tabla 3: Ejemplo del resultado de la consulta

host	count
fastlane.rubiconproject.com	100
ocsp.comodoca.com	72
ib.adnxs.com	56
ocsp.godaddy.com	48
secure.adnxs.com	46
ams1-ib.adnxs.com	30
beacon.krxd.net	30
tribune-d.openx.net	28
img.rtve.es	24

Ejercicio 4. Crear un script HIVE que obtenga la cantidad de 'UserAgent' distintos junto con el número de peticiones observadas en la traza de análisis. (0.75 puntos)

En este caso, tras añadir el campo *header_user-agent*, se hace un recuento de la cantidad de agentes distintos junto al recuento de todas las filas.

Código 4: Consulta para obtener el resúmen de agentes.

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SET net.ripe.hadoop.pcap.io.reader.class=
net.ripe.hadoop.pcap.HttpPcapReader;

SELECT COUNT(DISTINCT header-user-agent) AS uacount,
COUNT(*) AS trcount
FROM http;
```

En la tabla 4 se muestra el resultado.

Tabla 4: Ejemplo del resultado de la consulta

ua_count	tr_count
2	334698

Ejercicio 5. Crear un script HIVE que obtenga las diferentes líneas de respuesta HTTP junto con el número de veces que se han observado. (0,75 puntos)

Para resolver este ejercicio se hace un recuento de observaciones agregado por la respuesta http.

Código 5: Consulta para obtener el recuento de respuestas

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SET net.ripe.hadoop.pcap.io.reader.class=
net.ripe.hadoop.pcap.HttpPcapReader;

SELECT http_response,
         count(*) AS res_count
         FROM http
         GROUP BY http_response;
```

En la tabla 5 se muestran las primeras líneas del resultado.

Tabla 5: Ejemplo del resultado de la consulta

http_response	res_count
NULL	332400
HTTP/1.1 200 OK	1792
HTTP/1.1 302 Found	250
HTTP/1.1 302 Moved Temporarily	96
HTTP/1.1 204 No Content	60

Se observa un número de respuestas OK superior al resto, un resultado razonable si tenemos en cuenta que la traza se genera mediante un uso normal de internet.

Ejercicio 6. Crear un script HIVE que obtenga la popularidad de los nombre de dominio (por número de peticiones) en la traza de análisis. (0,75 puntos)

Para resolver este ejercicio nos basamos en la cantidad de requests DNS. Al ser un protocolo de resolución de nombres de dominio a direcciones, para encontrar los dominios más populares basta contar la cantidad de veces que se han resuelto dichos dominio por DNS.

En esta consulta los nombres de dominio se obtienen del primer substring (separando por espacios) de la columna *dns_request* y se hace un recuento agregando por esta columna calculada. No se ha encontrado una forma más sencilla y directa de hacerlo con el serde proporcionado.

Código 6: Consulta para obtener el ranking de dominios

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SET net.ripe.hadoop.pcap.io.reader.class=
net.ripe.hadoop.pcap.DnsPcapReader;

CREATE TABLE IF NOT EXISTS dns_domains  AS
select split(substring(dns_question, 0,
 instr(dns_question, ' ')),'\\.')[
SIZE(split(substring(dns_question, 0,
 instr(dns_question, ' ')),'\\.'))-3]
```

```
AS site_name_splitted FROM dns;

SELECT site_name_splitted,
count(*) AS counter
FROM dns_domains
GROUP BY site_name_splitted
ORDER BY counter DESC;
```

En la tabla 6 se muestra el top 5.

Tabla 6: Ejemplo del resultado de la consulta

dominio	nº visitas
google	388
doubleclick	384
adnxs	376
facebook	272
uam	264

Ejercicio 7. Obtener los valores por defecto de los siguientes parámetros (desde HIVE) e indicar cómo se han obtenido:(1 punto)

Los valores se han obtenido en HUE, ejecutando en el editor de HIVE el comando *SET –variable–*. En la tabla 7 se muestra el resultado.

Tabla 7: Resultado de la consulta

parámetro	valor
mapreduce.map.memory.mb	0
mapreduce.map.java.opts	-Djava.net.preferIPv4Stack=true
mapreduce.reduce.memory.mb	0
mapreduce.reduce.java.opts	-Djava.net.preferIPv4Stack=true
mapreduce.task.io.sort.mb	256
yarn.scheduler.minimum-allocation-mb	1024
yarn.scheduler.maximum-allocation-mb	65536
yarn.scheduler.minimum-allocation-vcores	1

Ejercicio 8. Utilizando el script desarrollado en el ejercicio 2, observar la influencia del parámetro "mapreduce.map.cpu.vcores". Variar

su valor entre 1 y 3. Realizar 4 ejecuciones con cada valor y obtener la media y al desviación estándar. Representar los datos de forma gráfica o mediante una tabla. (1,5 puntos)

En este ejercicio se ejecuta la consulta dos variando el parámetro del cluster mediante el comando *SET*.

Código 7: Consulta para obtener el registro de flujos (ampliado)

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

SET mapreduce.map.cpu.vcores=1;
-- o bien el valor 3

USE uambd11;

SELECT src,
dst,
protocol,
sum(len) AS bytes,
count(*) AS packages
FROM pcaps
WHERE protocol IN ('UDP', 'TCP')
GROUP BY src, dst, protocol;
```

En la tabla 8 se muestra el resultado.

Tabla 8: Resultado de la consulta (segundos)

Ejecución	mapreduce.map.cpu.vcores=1	mapreduce.map.cpu.vcores=3
1	48.4	47.06
2	46.03	48.36
3	47.89	44.82
4	8.22	48.606
std	1.09	1.73
media	47.63	47.21

No se observan diferencias significativas entre ambos valores del parámetro.

Ejercicio 9. Utilizando el script desarrollado en ejercicio 3, observar la influencia del parámetro "mapreduce.map.memory.mb". Probar con

los siguientes valores [2048,4096,8192] . Realizar 4 ejecuciones con cada valor y obtener la media y al desviación estándar. Representar los datos de forma gráfica o mediante una tabla. (1,5 puntos)

Código 8: Consulta para obtener el ranking de hosts (ampliado)

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SET net.ripe.hadoop.pcap.io.reader.class=
net.ripe.hadoop.pcap.HttpPcapReader;

SET mapreduce.map.memory.mb = 2048
--o bien 4096, 8192

SELECT header-host, count(http-request) as count
FROM http
GROUP BY header-host
ORDER BY rank DESC LIMIT 10;
```

En la tabla 9 se muestra el resultado.

Tabla 9: Resultado de la consulta (segundos)

Ejecución	.memory.mb = 2048	.memory.mb = 4096	.memory.mb = 8192
1	50.85	57.32	45.47
2	51.96	55.05	52.53
3	51.97	53.95	48.8
4	53.18	52.85	44.74
std	0.95	1.90	3.56
media	51.99	54.79	47.88

El parámetro controla la cantidad máxima de memoria que puede usar un mapper en Mb. Al incrementar la memoria el proceso, se observa que es ligeramente más rápido. Es posible que con una mayor cantidad de datos esta diferencia en tiempo de ejecución sea mayor.

Ejercicio 10. Utilizando el script desarrollado en ejercicio 6, observar

la influencia de la aplicación de la vectorización de HIVE (activar todos los parámetros de vectorización) . Realizar 4 ejecuciones con cada una de las dos opciones y obtener la media y al desviación estándar. Representar los datos de forma gráfica o mediante una tabla. (1,5 puntos)

En este ejercicio se deben activar las funcionalidades de vectorización y guardar la tabla necesaria para la query en un formato que soporte vectorización (en este caso ORC). La vectorización permite operar bloques de filas (1024 filas) en una misma iteración lo que disminuye el uso de recursos de la CPU [1]. La funcionalidad se activa con el parámetro de HIVE: *hive.vectorized.execution.enabled*.

Código 9: Consulta para obtener el ranking de dominios

```
ADD JAR hdfs:///user/uambd11/libs/
hadoop-pcap-serde-1.2-
SNAPSHOT-jar-with-dependencies.jar;

USE uambd11;

SET net.ripe.hadoop.pcap.io.reader.class=
net.ripe.hadoop.pcap.DnsPcapReader;
hive.vectorized.execution.enabled=true
SET hive.vectorized.execution.enabled=true;

CREATE TABLE IF NOT EXISTS dns_domains_vorc
STORED AS ORC -- necesario para vectorizar
AS SELECT split(substring(dns_question, 0,
instr(dns_question, ' ')),'\\.')[
SIZE(split(substring(dns_question, 0,
instr(dns_question, ' ')),'\\.'))-3]
AS site_name_splitted FROM dns;

SELECT site_name_splitted,
count(*) AS counter
FROM dns_domains_vorc
GROUP BY site_name_splitted
ORDER BY counter DESC;
```

En la tabla 10 se muestra el resumen de tiempos.

[1] https://cwiki.apache.org/confluence/display/Hive/Vectorized+Query+Execution

Tabla 10: Resultado de la consulta (segundos)

Ejecución	vectorizado	sin vectorizar
1	38.01	40.15
2	38.04	40.33
3	37.91	42.24
4	39.51	40.13
std	0.76	1.02
media	38.36	40.71

Se aprecia una ligera bajada del tiempo de ejecución al vectorizar la operación, aunque no es significativa.