



TAREA 1 BIG DATA



20 DE AGOSTO DE 2017
UNIVERSIDAD DE LOS ANDES
María José González

Consultas sobre base de datos (MongoDB)

Tweets totales: 963968, 2,005GB.

1. Cantidad de tweets retweeteados: Obtuve la cantidad de tweets subiendo de 100⁹ unidades, mediante el comando:

```
> db.icccualquiercosa.count({"retweeted_status.retweet_count": {$gt: 1000}})
289339
> db.icccualquiercosa.count({"retweeted_status.retweet_count": {$gt: 2000}})
230358
> db.icccualquiercosa.count({"retweeted_status.retweet_count": {$gt: 3000}})
195217
> db.icccualquiercosa.count({"retweeted_status.retweet_count": {$gt: 4000}})
173515
> db.icccualquiercosa.count({"retweeted_status.retweet_count": {$gt: 5000}})
157954
> db.icccualquiercosa.count({"retweeted_status.retweet_count": {$gt: 20000}})
60037
>
```

Tiempo	Retweets
34	1000
30	2000
38	3000
45	4000
36	5000
37	20000






















2. Creación de Tweets: se obtuvo la cantidad de tweets por rango de un año:

```
> db.icccualquiercosa.count({"created_at": {$gt: "Mon May 20 00:00:00 +0000 2016", $lt: "Sun May 30 00:00:00 +0000 2017"}})
910346
```

3. Tweets por ciudad:

```
> db.icccualquiercosa.count({"user.location": "New York"})
2509
> db.icccualquiercosa.count({"user.location": "Boston"})
744
> db.icccualquiercosa.count({"user.location": "Atlanta"})
695
> db.icccualquiercosa.count({"user.location": "Washington"})
256
> db.icccualquiercosa.count({"user.location": "Los Angeles"})
2595
```

Estado del sistema sin consultas:

Monitor del sistema					
Procesos		Recursos		Sistemas de archivos	
Nombre del proceso	Usuario	% CPU ▲	ID	Memoria	Prioridad
 compiz	mjgonzalez6	30	1686	196,6 MiB	Normal
 gnome-system-monitor	mjgonzalez6	4	28021	46,6 MiB	Normal
 firefox	mjgonzalez6	1	2252	581,3 MiB	Normal
 atom	mjgonzalez6	0	21491	32,4 MiB	Normal
 atom	mjgonzalez6	0	21346	54,1 MiB	Normal
 atom -type=gpu-process -cha	mjgonzalez6	0	21374	2,6 MiB	Normal
 atom -type=renderer -no-san	mjgonzalez6	0	21387	138,9 MiB	Normal
 atom -type=zygote -no-sandb	mjgonzalez6	0	21350	2,2 MiB	Normal
 at-spi2-registryd	mjgonzalez6	0	1752	304,0 KiB	Normal
 at-spi-bus-launcher	mjgonzalez6	0	1685	320,0 KiB	Normal
 bamfdaemon	mjgonzalez6	0	1658	30,5 MiB	Normal
 bash	mjgonzalez6	0	25993	5,9 MiB	Normal
 bash	mjgonzalez6	0	27125	5,9 MiB	Normal
 dbus-daemon	mjgonzalez6	0	1570	1,3 MiB	Normal
 dbus-daemon	mjgonzalez6	0	1705	336,0 KiB	Normal
 dconf-service	mjgonzalez6	0	1743	460,0 KiB	Normal
 deja-dup-monitor	mjgonzalez6	0	2240	520,0 KiB	Normal
 evolution-addressbook-factor	mjgonzalez6	0	2028	N/D	Normal
 evolution-addressbook-factor	mjgonzalez6	0	2048	308,0 KiB	Normal
 evolution-calendar-factory	mjgonzalez6	0	1904	N/D	Normal
 evolution-calendar-factory	mjgonzalez6	0	2015	2,4 MiB	Normal

Estado del sistema con consultas:

Monitor del sistema					
Procesos		Recursos		Sistemas de archivos	
Nombre del proceso	Usuario	% CPU ▲	ID	Memoria	Prioridad
compiz	mjgonzalez6	23	1686	196,6 MiB	Normal
gnome-system-monitor	mjgonzalez6	19	28021	46,6 MiB	Normal
firefox	mjgonzalez6	4	2252	585,4 MiB	Normal
gnome-terminal-server	mjgonzalez6	0	25986	16,8 MiB	Normal
atom	mjgonzalez6	0	21491	32,4 MiB	Normal
atom	mjgonzalez6	0	21346	54,0 MiB	Normal
atom -type=gpu-process -cha	mjgonzalez6	0	21374	2,6 MiB	Normal
atom -type=renderer -no-san	mjgonzalez6	0	21387	137,4 MiB	Normal
atom -type=zygote -no-sandb	mjgonzalez6	0	21350	2,2 MiB	Normal
at-spi2-registr	mjgonzalez6	0	1752	304,0 KiB	Normal
at-spi-bus-launcher	mjgonzalez6	0	1685	320,0 KiB	Normal
bamfdaemon	mjgonzalez6	0	1658	30,5 MiB	Normal
bash	mjgonzalez6	0	25993	5,9 MiB	Normal
bash	mjgonzalez6	0	27125	5,9 MiB	Normal
dbus-daemon	mjgonzalez6	0	1570	1,3 MiB	Normal
dbus-daemon	mjgonzalez6	0	1705	336,0 KiB	Normal
dconf-service	mjgonzalez6	0	1743	460,0 KiB	Normal
deja-dup-monitor	mjgonzalez6	0	2240	520,0 KiB	Normal
evolution-addressbook-facto	mjgonzalez6	0	2028	N/D	Normal
evolution-addressbook-facto	mjgonzalez6	0	2048	308,0 KiB	Normal
evolution-calendar-factory	mjgonzalez6	0	1994	N/D	Normal

Índice de Mongo

Configuramos un índice de Mongo:

```
> db.icccualquiercosa.ensureIndex({"_id":1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 1,
  "note" : "all indexes already exist",
  "ok" : 1
}
```

Configuramos un índice para la cantidad de retweets:

```
> db.icccualquiercosa.ensureIndex({"_id":1,"retweeted_status.retweet_count": -1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

En resumen, se demora al menos 10 segundos menos en la ejecución.

Problemas y dificultades

- Si bien pude exportar los datos de MongoDB a un archivo cvs, no pude importarlo a una base de datos de SQL.

Empresa Tecnológica: Twitter

Principios de Desarrollo de Software

1. *Tolerancia a fallas:* cuando ocurre algún evento de comunicación, se busca reducir el impacto que este pueda tener en el dispositivo donde se ejecuta la aplicación, tanto en su rendimiento de batería como en el uso del ancho de banda que solicita el evento para su realización. Para llevar a cabo esta tarea, los eventos se envían en batches y se comprimen antes de ser enviados. Para asegurar que los datos siempre lleguen a los servidores de Twitter, los dispositivos reintentan las transferencias fallidas de datos hasta un límite de tamaño en disco en el dispositivo.
2. *Baja Latencia:* por otro lado, para que los eventos gatillados lleguen a los servidores tan pronto como sea posible, existen varios triggers que provocan que el dispositivo intente una nueva transferencia: un trigger de tiempo que se activa cada pocos minutos cuando la aplicación se pone en primer plano.
3. *Escalabilidad:* el protocolo de comunicación existente en Twitter resulta en muchos dispositivos enviando cientos de miles de cargas comprimidas cada segundo. Cada una de



esas cargas pueden contener cientos de eventos. Para permitir que esta carga sea fácilmente llevada a cabo con un escalamiento lineal, se emplea un servicio escrito en GOLANG, provisto por Amazon Elastic Load Balancer (ELB), y simplemente posiciona cada carga en una cola que recibe en una Kafka queue.

4. *Extensibilidad:* dado que Kafka guarda los mensajes que escribe en disco y soporta múltiples copias de cada mensaje, es un almacenamiento durable y estable. Por lo tanto, una vez que la información está contenida en ella, se pueden tolerar retrasos o fallos por procesamiento o reprocesamiento de datos posteriormente. Sin embargo, Kafka no es la verdadera fuente de los datos históricos dado que se necesitarían cientos de bases de datos para almacenar todos estos datos solo por unos días. Por lo tanto, se configuró el cluster de Kafka para retener información por algunas horas (tiempo suficiente para responder a fallos

inesperados) y traer la data de los almacenamientos permanentes, Amazon Simple Storage Service (Amazon S3) tan pronto como sea posible. Twitter utiliza Storm para el procesamiento de datos en tiempo real, como lo es primeramente leer la información desde Kafka y almacenarla dentro de Amazon S3.



Arquitectura Lambda

1. *Batch Layer*: cuando los datos son almacenados dentro de Amazon S3, estos deben estar dispuestos para poder calcular cualquier cosa que estos datos pueda proveer por medio de Amazon Elastic MapReduce (Amazon EMR). Esto incluye batch Jobs para todos los datos que un usuario puede observar en su dashboard, así como demos de las características en las que se está trabajando actualmente.



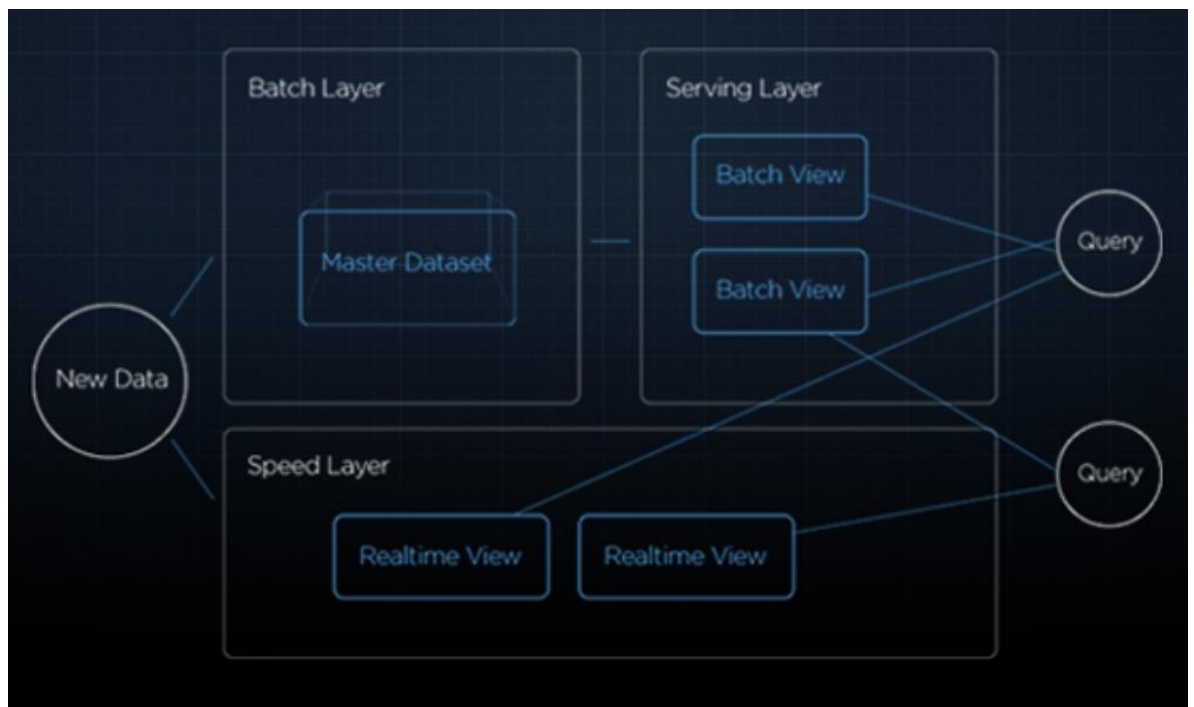
Twitter elabora su MapReduce en Cascading y lo ejecuta vía Amazon EMR. Amazon EMR lee los datos que está almacenada en Amazon S3 como input y guarda el resultado de vuelta a Amazon S3 una vez que el procesamiento se completa. Esto se detecta gracias a una topología de scheduler ejecutándose en Storm y depositando el output de Amazon S3 dentro de un cluster e Cassandra para que esté disponible para hacer operaciones de querying.

2. *Speed Layer*: lo anterior describe un framework durable y tolerante a errores para llevar a cabo computaciones analíticas. Sin embargo, esto presenta un problema: no es ejecutado en tiempo real. Algunas de estas operaciones se realizan por hora, mientras que otras ocupan un día completo para procesar los datos de input. El rango de tiempo de computación puede fluctuar entre horas y minutos, como lo hace el tiempo que toma obtener el output desde Amazon S3 hacia la Serving Layer. Así, en el mejor de los casos, los datos estarían siempre con unas pocas horas de retraso y no cumplirían los objetivos de ser en tiempo real y accionables.
Para abordar esto, se realizan cálculos de flujo con el archivo que llega a los servidores.



Una topología independiente de Storm computa el mismo topic que Kafka y realiza los mismos cálculos que MapReduce, pero en tiempo real. El output de estos cálculos se guardan en un cluster diferente independiente de Cassandra para las consultas en tiempo real. Para compensar el hecho de que hay menos tiempo para ejecutar esto, y potencialmente menos recursos, tanto en la Speed Layer como en la Batch Layer se usan algoritmos probabilísticos como Bloom Filters y HyperLogLog (así como algunos desarrollados por la misma Twitter). Estos algoritmos permiten optimizar el espacio y el tiempo sobre alternativas de fuerza bruta, con el costo de insignificante pérdida de precisión.

Combinando las capas



Como las computaciones en la capa Batch son repetibles y tienen mayor tolerancia a fallos que la capa Speed, la API siempre favorece la capa Batch para procesar los datos.

Bibliografía:

Artículo oficial de Twitter para el análisis de la empresa tecnológica:

https://blog.twitter.com/engineering/en_us/a/2015/handling-five-billion-sessions-a-day-in-real-time.html

Página web de la Nasa para descargar datos mediante un script de Python:

<https://wiki.earthdata.nasa.gov/display/EL/How+To+Access+Data+With+Python>