Big Data

Trabajo Práctico 2

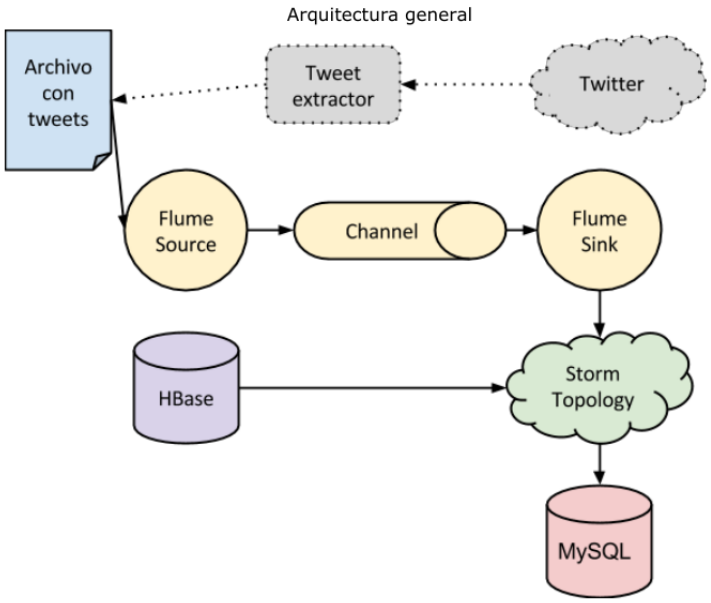
Crespo, Alvaro	50758
Petit, Alejandro	48308
Susnisky, Dario	50592
Videla, Máximo	51071

18 de noviembre de 2013

1. Presentación del problema

El objetivo de este trabajo práctivo es el de implementar un proyecto que realice un procesamiento en tiempo real de datos obtenidos de *Twitter*. Los *tweets* se van descargando mediante un cliente y son procesados a medida que llegan. Para esto se utiliza *Flume* detectando cuando llegan nuevos *tweets*. Una vez encolados los *tweets* por *Flume*, *Storm* es el encargado de procesarlos, implementando las métricas deseadas. Para este último punto, *Storm* toma ciertos datos de *HBase* y finalmente vuelca los resultados de las métricas en una tabla en *MySQL*.

A continuación se presenta un esquema general de la aplicación:



2. Decisiones de implementación

2.1. Agente de Flume

Con respecto al agente de Flume se implementó un simple agente con un *source*, 3 *sinks* y 3 *channels*, uno que vincule cada *sink* con el *source*.

Para el *source* se utilizó un *Spooling Directory Source*, que viene dado en el paquete de Flume, que simplemente se queda observando un determinado directorio y cuando se agregan nuevos archivos, genera un evento nuevo por cada línea. En nuestro caso, cada línea es un objeto JSON con la información de un tweet.

La decisión de tener varios *sinks* vino por diferentes razones. En primer lugar, se utilizaron el *Logger Sink* y el *File Roll Sink*, que dirigen el output del agente de Flume a consola (mediante logs de debugging de *Log4j*) y a archivos de texto en un directorio de salida, respectivamente. Esto tenía el objetivo principal de familiarizarnos con la nueva tecnología y a la vez ver el funcionamiento del Spooling Directory Source para nuestras necesidades. Pero luego también decidimos continuar utilizandolas para testear y verificar el correcto funcionamiento del *ActiveMQ Sink* que implementamos.

2.2. Topología de Storm

Al implementar una topología en Storm que permita medir nuestra métrica desarrollamos el siguiente modelo:

- Un *Spout* encargado de tomar los datos de la cola de mensajes.
- Un *Bolt* que recibe todos los mensajes que envía el *Spout* sin importar el orden. A su vez este *Bolt* toma de *HBase* todos los grupos de *keywords* existentes y emite mensajes indicando cuántas menciones por grupo recibe por *tweet*.
- Un último *Bolt* que se encarga de sumar todos los mensajes que recibe y guardar la cantidad de menciones de cada grupo en una base de datos *MySQL*.

El *Spout* está unido al primer *Bolt* mediante *ShuffleGrouping* ya que no importa el orden de los mismos. El primer *Bolt* se encuentra unido al segundo con *FilterGrouping* agrupando por el *GroupId* ya que solamente un *Bolt* debe ser encargado de actualizar un element (en el ejemplo, uno de los partidos políticos) en la base de datos.

Para probar la topología se implementaron *Bolts* que hacen sus tareas omitiendo las conexiones a *MySQL* y *HBase*. Habiendo corroborado su correcto funcionamiento se implementaron luego, las versiones finales y completas.

Un gran problema que tuvimos, y con el cual estuvimos luchando hasta último momento, fue que, por querer mantener solo una conexión con la base de datos *MYSQL* por *Bolt*, almacenábamos dicha conexión como una variable de instancia, con el grave error de ponerle un calificador *transient*, pensando (de manera errónea) que esto salvaría los problemas de serialización. El error nos confundió y nos tomó bastante trabajo encontrarlo, principalmente porque al correr el sistema de forma local, todo funcionaba correctamente, pero al correr en el cluster, algo claramente fallaba luego de desencolar los mensajes dado que nunca se escribían los resultados a la base de datos. Este error nuestro se vió magnificado por algunas de las limitaciones de storm, los archivos de logs están distribuidos en el cluster, con lo cual no podíamos fácilmente acceder a ellos. Además, la storm UI no siempre evidenciaba la *Null Pointer Exception* que hacía que los bolts que escribían a la base datos, y por lo tanto la topología, fallaran en su cometido.

2.3. Cliente de Twitter

Twitter cuenta con 2 *APIs* para consultar información. La primera es *REST*, dandole a un desarrollador métodos para realizar pedidos HTTP apropiados, que contengan como respuesta lo que se esta buscando. Por otro lado, *Twitter* cuenta con una *API* de *streams* que encola *tweets* (u otros eventos) constantemente y con bajo costo, evitando el *overhead* de la *REST API*.

Dada la naturaleza y los requerimientos de la aplicación, fue casi intuitiva la decisión de utilizar la segunda *API*. Contando con esto y con la decisión de desarrollar la aplicación en *Java*, se dicidió utilizar el cliente *Hosebird* dado que proveía los servicios que buscabamos para nuestra aplicación (acceso a la *Streaming API*, autenticación vía *Oauth* y otros). Además, *Hosebird* permite integración sencilla con *Maven* y nos dió la serguridad de tener una prueba funcional en poco tiempo.

3. Problemas encontrados

3.1. Agente de Flume

El principal problema, o desafío, de la implementación del agente de Flume fue la necesidad de implementar un *Sink* custom, para interactuar con la cola de *AciveMQ*, ya el tanto para el *Souce* como para los *Channels*, utilizamos implementaciones como Spooling Directory Source o *Memory Channel*, que ya vienen provistas por Flume.

Un problema, o una lección, que tuvimos con Flume fue que si bien es posible que un *source* se conecte a varios *channels*, no sucede lo mismo con los *sinks*. Un *sink* puede estar conectado a un y sólo un *channel*. Además, al configurar el *source* de nuestro agente con varios *channels*, lo que se denomina *Fan Out Flow*, nos topamos con las diferentes formas en que un source puede estar configurado para manejar conexiones a varios *channels*, o la "política de Fan Out". Existen dos formas o políticas para esto: replicar o multiplexar. En el primer caso, el *source* le manda el evento generado a todos los *channels* conectados, mientrás que en el segundo, el evento se envía a un subconjunto de ellos. En nuestro caso, nos topamos con un error al configurar varios *channels* ya que, al parecer, el default en estos casos es multiplexar, lo cual no era lo que necesitabamos.

3.2. Topología de Storm

La configuración de *Storm* para encontrar *activeMQ* en modo distribuido nos presentó un desafío importante. Si bien contabamos con la configuración local, al comenzar a probar la aplicación en modo distribuido nos encontramos diversas complicaciones.

Por otra parte, al implementar la topología, no fueron triviales las conexiones entre *Spouts* y *Bolts* ya que dependiendo del caso, requerían agrupaciones diferentes. Finalmente logramos encontrar agrupaciones que permitiesen llegar a nuestro objetivo.

3.3. Cliente de Twitter

Si bien en la versión final del proyecto se utilizó *Hosebird client*, en un comienzo se intentó utilizar la librería *Twitter4J*. En un principio, esta librería, parecía una potente posibilidad para consultar la *Twitter API*. Finalmente se decidió utilizar *Hosebird client* ya que provee una mejor documentación y un enfoque completo en la *Streaming API* en vez de la *REST API* (que es la que finalmente se decidió usar).

4. Instrucciones para ejecutar la métrica

El proyecto cuenta con 3 partes que deben ejecutarse de manera paralela. Se considera para la compilación de las 3 partes, que *Maven* se encuentra instalado en el sistema.

4.1. Cliente de Twitter

4.1.1. Compilación

Para compilar el proyecto del consumidor de streams de *Twitter* correr dentro de la carpeta tweetsStream:

mvn clean package

Para que el proyecto se compile de forma correcta, en la carpeta /tweetsStream/src/-main/resources debe existir el archivo oauthcredentials con el siguiente formato:

consumerKey = xxxxxxxx
consumerSecret = xxxxxxx
accessToken = xxxxxxx
accessTokenSecret = xxxxxxxx

4.1.2. Ejecución

Para correr el consumidor de streams de *Twitter* correr dentro de la carpeta tweetsS-tream/target:

java -jar bigdata-twitterclient-jar-with-dependencies.jar <outputFolder><terms>

El parámetro <outputFolder>es el path relativo de la carpeta en donde se crearan los archivos de salida. Esta carpeta debe existir previamente. Paralelamente <terms>es la lista de términos a buscar separados por comas (","). Un ejemplo de términos podría ser "termino1,termino2,termino de prueba".

4.2. Flume

Se requiere que *Flume* se encuentre instalado y agregado a la variable de entorno PATH. Esto último se puede hacer de la siguiente manera:

export PATH=absolute-path/to/flumeDir/bin:$PATH

Para verificar que *Flume* está agregado al PATH se debería poder correr desde la consola el comando “flume-ng“ y verificar que el output es el del comando en cuestión.

4.2.1. Compilación

Para compilar el proyecto *Flume* con el custom sink para activeMQ correr dentro de la carpeta flume

mvn clean package -DskipTests

4.2.2. Ejecución

Para correr el agente de *Flume* correr dentro de la carpeta flume

sh run-flume.sh

La configuración del agente de *Flume* se encuentra en la carpeta flume/conf.

4.3. Storm

4.3.1. Compilación

Para compilar el proyecto *Storm* con la topología correr dentro de la carpeta storm

mvn clean package

4.3.2. Ejecución

Para correr la topología de *Storm* correr dentro de la carpeta storm

storm jar target/storm-0.0.1-SNAPSHOT-jar-with-dependencies.jar com.itba.g2.storm.StormTopology topologyName sqlDBUrl table mainField countField user pass columnFamily

Los parámetros son los siguientes:

- "topologyName" es el nombre que se le quiere dar a la topología de storm que se correrá.
- "activeMQUrl" es la url de la cola de mensajes *ActiveMQ*.
- "sqlDBUrl" es la url de la base de datos *MySQL* donde se escribirán los resultados.
- "table" es el nombre de la tabla donde se almacenarán los resultados.
- "mainField" es el campo de la tabla que representa el nombre de la agrupación.
- "countField" es el campo de la tabla que representa la cantidad de menciones.
- "user" es el usuario para acceder a la base de datos *MySQL*.
- "pass" es la usuario para acceder a la base de datos *MySQL*.
- "HBaseTable" es el nombre de la tabla de *HBase* las palabras a consumir.
- "columnFamily" es el nombre de la columna de la tabla de *HBase* que contiene las palabras a buscar.

Una vez ejecutada una topología, si se desea volverla a correrla (con el mismo nombre) se debe dar de baja previamente la topología anterior. Esto se logra corriendo

storm kill topologyName
En caso de que Nimbus se encuentre caído, lo cual puede percibirse desde la storm UI o por algunos errores extraños de conexión, se debe levantarlo corriendo

storm nimbus Adicionalmente, quizás la storm UI puede estar desactivada, para activarla basta correr
storm ui Si se encuentra bajo un proxy, como es el caso de los laboratorios de ITBA, que bloquea ciertos puertos, puede tener problemas para monitorear algunos los sistemas. Esto puede solucionarse abriendo un tunel ssh al namenode.

ssh -i id_dsa -L 9000:hadoop-2013-datanode-4:8080 hadoop@hadoop-2013-namenode
Por último, en caso de que no haya supervisors corriendo se deben levantar de la siguiente forma corriendo en el cluster

nohup storm supervisor&

4.4. Consideraciones

4.4.1. ActiveMQ

Tanto el proyecto *Flume* como el de *Storm* utilizan una cola de mensaje *ActiveMQ*, que debe estar correctamente instalado, y cuya url es uno de los parámetros para correr el proyecto de *Storm*.

Para monitorear el funcionamiento de las colas en *ActiveMQ* se puede acceder a http://hadoop-2013-datanode-7:8161/admin/queues.jsp , en donde, al tiempo de este proyecto se encuentra levantaba la cola de mensajes de *ActiveMQ*.

Para monitorear el funcionamiento de la topología de *Storm*, se puede acceder a la storm ui en http://hadoop-2013-datanode-4:8080.

5. Formato de los resultado de la métrica

Los resultados de la métrica implementada se almacenan en una tabla de MYSQL, conforme a los requerimientos de la cátedra. El esquema que definimos para esta tabla consiste simplemente de los campos:

- GroupId: El nombre del grupo o conjunto de palabras de interés
- Ammount: La cantidad de menciones de palabras relacionadas con el grupo