Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

Social Data Aggregator User Guide

Project full title: Future Internet Core
Project acronym: FI-Core
Contract No.: 632893

Contents

Introduction

Welcome the User and Programmer Guide for the Social Data Aggregator Generic Enabler. The online documents are being continuously updated and improved, and so will be the most appropriate place to get the most up to date information on using this interface.

OVERVIEW

Over time all generations have come to embrace the changes social network has brought about. Nowadays online social media have gained astounding worldwide growth and popularity. They are generating a huge amount of records about users' activities and also attracting attention from variety of researchers and companies worldwide. Every day data collected by social networks are gathered and feed a variety of analytics (users behavior, habits, topic trends..) which are capable of extracting significant patterns and further analisys. The aim of Social Data Aggregator (SDA) GE is to retrieve data from different Social Networks and provide different analytics depending on user needs. The GE relies on Apache Spark for computation on data. The SDA comes with different built-in modules but custom modules can be added as well.

Source

The source code of this project can be cloned from the GitHub Repository. Code for related libs can be found on GitHub FiwareTIConsoft Group.

FIWARE Social Data Aggregator - User Guide

INSTALLATION

To deploy Social Data Aggregator from source go to the project main folder and launch the following command:

`mvn clean package`

Once built SocialDataAggregator with Maven, under the folder scripts/your_os_env/ there is a script called make-dist. Run it with the following syntax (e.g. linux):

`./make-dist.sh <output_folder>`

where _output_folder is the folder inside which you want to create the SDA GE folder tree. When the script finish to run, you should see the following dir tree:

```
sda
 -  bin (contains all the binaries of sda in their respective folder)
 - confs (contains the configurations of each specific sub-component)
 - scripts (contains the launch scripts for each sub-component and a start-all script to start a
    ll components)
```

SETTING UP THE START-ALL/START-ENV SCRIPTS:

This script launches all the modules of the real-time part of the social data aggregator.

Under the /scripts folder there are two files:

- *confs.cfg.template.* Remove the .template extension and edit the file providing:

Property	Description
SPARK_HOME	the location of your spark installation
SPARK_MASTER_WEBUI _PORT	port for the web ui
SPARK_MASTER_IP	IP of spark master
SPARK_MASTER_PORT	spark master port
SPARK_WORKER_INSTA NCES	number of worker instance per node (default 3)
SDA_HOME	(optional) Home of SDA (otherwise will be guessed by the script from the location of the start-all script)

- *modules:* This file contains all the modules that will be started from the start-all script. Add a comment (#) on the modules you don't need to avoid starting them.

The script can be ran in two ways:

1) submitting the applications on an existent spark cluster:

```
./start-all.sh
```

1) setting up a spark cluster in standalone mode before submitting the applications

```
./start-all.sh --start-spark-env
```

In both cases you need to edit the configuration file (in the first case to refer to the already existent master, in the second to know with which configurations to deploy it).

CONNECTORS

Connectors are in charge to retrieve the data from Social Networks (SNs). Each connector is "specialized": it is connected with a specific social network and gathers data by interacting with it. The way data are retrieved can vary from SN to SN: some SNs provide stream Apis (e.g. twitter,instagram) while others that needs to be polled by the connector. The connector receives live input data streams and divides the data into batches namely a specific set of data collected during a given timeframe. The content of each batch is mapped onto an internal model and then sent to the internal bus to make it available to real time consumers. The internal stream bus is a communication bus for a loosely-coupled interconnection between modules. It can be implemented by different technologies (apache kafka, amazon kinesis, rabbitMQ..). Data belonging to different batches are also collected in a window. The content of each window is saved on the storage as raw data in json format. In this way raw data can be subsequently processed by batch consumers. The storage has to be reachable from every node of the cluster, it can be implemented by a Database (Mysql, OrientDB, MongoDB..), a distributed filesystem (HDFS..), an online file storage web service (s3) or a shared disk (NFS).

Each connector can expose apis that can be contacted from a *controller* in order to modify the settings or the topics being under monitoring. A topic can be based on:

- key-word(s)
- geo location (latitute,longitude..)
- a target user (if the social network allows user tracking)
- hashtags

SETTING UP CONNECTOR-TW

CONFIGURATIONS

Under the folder *sda\confs\connector-tw* you will find 3 configuration files:

log4j.properties

the properties for log4j. Set where you want the connector log. Edit this file following your needs.

twstats.cfg.xml

configuration file for hibernate. Edit it if you compiled the GE with the DAO default implementation. If you provide a different implementation you can leave this file as is or delete it. Edit the following fields with your database configuration:

```
<property name="connection.url"></property>
<property name="connection.username"> </property>
<property name="connection.password"> </property>
```

You can find the model of the default DAO in social-data-aggregator/data_model in the project directory.

TwStreamConnector.properties

Key Name	Optional	Description

Twitter Configurations

In this section of the configuration file there are all the properties regarding the connection with Twitter:

Key Name	Optional	Description
twConsumerKey	NO	Consumer Key of the twitter application
twConsumerSecret	NO	Consumer Secret of the twitter application
twToken	NO	User token
twTokenSecret	NO	User token secret

Node Configurations

In this section of the configuration file there are the configurations regarding the node that hosts the driver:

Key Name	Optional	Description
nodeName	NO	The name of the node (the value must be the same of the field monitoring_from_node in the db model in case you use the default DAO). This property is needed in case of multiple instances of the collector in nodes that have different Public IPs but share the same rdbms. In this way you can choose which key will be monitored from a target node.
proxyPort	YES	(Uncomment this property in case you use a proxy for outbound connections) The proxy port
proxyHost	YES	(Uncomment this property in case you use a proxy for outbound connections) The proxy host

Spark Configurations

In this section of the configuration file there are the configurations regarding the spark Streaming Context:

Key Name	Optional	Description
numMaxCore	YES	Number of cores to associate to this application (in case you have to run multiple streaming application) If you run just the collector you can comment this property
checkpointDir	NO	Directory where spark will save this application checkpoints
sparkBatchDuration Millis	NO	Duration of the batch (in milliseconds). It is the basic interval at which the system with receive the data in batches
sparkCleanTTL	NO	Duration (seconds) of how long Spark will remember any metadata (stages generated, tasks generated, etc.). Periodic cleanups will ensure that metadata older than this duration will be forgotten.
twitterInserterWindow Duration	NO	Duration of the window. Both the window duration and the slide duration must be multiples of the batch interval. Save frequency for gathered data.
twitterInserterWindow SlidingInterval	NO	Window sliding interval. The interval at which the window will slide or move forward. (set equal to the twitterInserterWindowDuration to avoid duplicated data saved)

App Configurations

In this section of the configuration file there are the configurations regarding the app:

Key Name	Optional	Description
serverPort	NO	The port on which jetty server will listen. Needed to start,restart,stop the collector.
savePartitions	NO	Number of partition to coalesce before save. Equals one will generate one file containing raw tweets for window.
dataOutputFolder	NO	the folder where the raw data will be saved
dataRootFolder	NO	Root folder on which data will be saved. Example: dataOutputFolder=<file://tmp/data> and dataRootFolder=raw will save data on ...
daoClass	YES	class for the custom dao if you don't want to use the default one

Kafka Configurations

In this section of the configuration file there are the configurations regarding the kafka. If you don't want the data sent on kafka delete or comment the following properties:

Key Name	Optional	Description
brokersList	NO	Kafka brokers list (separated by ,)
kafkaSerializationClass	NO	Default **kafka.serializer.StringEncoder** Change it if you want another serializer.
kafkaRequiredAcks	NO	tells Kafka the number of acks you want your Producer to require from the Broker that the message was received.
maxTotalConnections	NO	number of total connections for the connection pool
maxIdleConnections	NO	number of idle connections for the connection pool
customProducerFactory Impl	YES	uncomment if needed other implementation for connection to bus different than kafka

CONSUMERS

Consumers are modules that retrieve from the storage raw data collected by the connectors or from the internal stream bus and produces different kind of analytics from gathered data.

Examples of analytics provided from the Social Data Aggregator are:

- **Basic Aggregations:** calculation of the ppm (posts per minute) or number of posts in a time range, grouped by keywords or belonging to specific geo located areas, to recognize trending topics (consumer-tw-tot).
- **Gender Recognition:** this feature is useful for social networks that don't provide information about the gender of the user (twitter for example). Recognizing a user gender from his profile is a challenging task.
- **Sentiment Analysis:** sentiment analysis aims to determine the attitude of a commenter upon a specific topic. It is used by the SDA to infere the mood of users with respect to a monitored topic.

By subscribing to a target topic and looking for a particular key, consumers can retrieve only the information that they really need, discarding any data when not relevant to their analytics. Result data can be saved on storage rather then re-injected to the internal bus to be processed from other consumers capable of other types of analytics.

CONSUMER TW-TOW

OVERVIEW

The consumer tw tot provide a count on tweets, retweets, reply on geo and hashtags based criteria for a user defined time interval.

There are two versions of this module:

- Stream
- Batch

Configuration

The confs/consumers/consumer-tw-tot folder contains the following files:

dao_impl.conf

A properties file with the properties needed from the ConsumerTwTotDao implementation. If you use the **ConsumerTwTotDaoDefaultImpl** you can leave this file blank. log4j.properties the properties for log4j. Set where you want the connector log. Edit this file following your needs.

twstats-tot-tw.cfg.xml

configuration file for hibernate. Edit it if you compiled the GE with the ConsumerTwTotDao default implementation. If you provide a different implementation you can leave this file as is or delete it.

Edit the following fields with your database configuration:

```
<property name="connection.url"></property>
<property name="connection.username"> </property>
<property name="connection.password"> </property>
```

You can find the sql code to create the consumer-tw-tot tables needed to store analytics result in social-data-aggregator/data_model in the project directory.

bus_impl.conf

This is the configuration file for the internal bus. By default is filled with apache Kafka configurations. If you want to use a different implementation please follow these steps:

1) Create a Java class that implements the BusConnection interface
2) Set the properties you need for your implementation into the bus_impl.conf file
3) Put the path to your implementation as the value for the property busConnImplClass into the **TwTot-ConsumerProps.properties** file (e.g “com.mypackage.MyImplClass”)

TwTotConsumerProps.properties:

COMMONS CONFIGURATIONS:

Key Name	Optional	Description
roundPos	NO	Decimal position on which round the latitude and longitude provided in case of geoLoc tweet (i.e. roundPos=3 , latitude=17.87654 -> latitude=17.876)
daoImplClass	NO	Java class that implements the ConsumerTwTotDao interface for the connection to the storage (default value: com.tilab.ca.sda.consumer.tw.tot.dao.ConsumerTwTotDaoDefaultImpl)

BATCH CONFIGURATIONS:

Key Name	Optional	Description
defaultInputDataPath	NO	Default Folder (on distributed filesystem) that contains input data for the batch app (can be override from the command line)
minPartitions	YES	Min number of partitions for the input file (default 1)

STREAM CONFIGURATIONS:

Key Name	Optional	Description
keyHt	NO	topic key for statuses containing hashTags (Default ht)
keyGeo	NO	topic key for statuses with geo location (Default geo)
defaultRoundMode	NO	On which time field round to group values (allowed values for this property are: min,hour,day)
granMin	YES	**Valid only if round mode is min.** Granularity, if you want to group tweets in minute intervals (e.g gran=5 will group by 5 minutes -> the number of tweets in 5 minutes)
numMaxCore	NO	Number of cores to associate to this application (in case you have to run multiple streaming application)
checkpointDir	NO	Directory where spark will save this application checkpoints
sparkBatchDurationMillis	NO	Duration of the batch (in milliseconds). It is the basic interval at which the system with receive the data in batches
sparkCleanTTL	NO	Duration (seconds) of how long Spark will remember any metadata (stages generated, tasks generated, etc.). Periodic cleanups will ensure that metadata older than this duration will be forgotten.
twTotWindowDurationMillis	NO	Duration of the window. Both the window duration and the slide duration must be multiples of the batch interval. Data window on which analysis will be made.
twTotWindowSlidingIntervalMillis	NO	Window sliding interval. The interval at which the window will slide or move forward. (set equal to the twTotWindowDurationMillis to avoid unexpected behaviour)
busConnImplClass	NO	Java class that implements the BusConnection interface for the interconnection with the internal stream bus (default: com.tilab.ca.sda.consumer.utils.stream.BusConnectionKafkaImpl)

DEPLOY

STREAM DEPLOY:

To deploy consumer-tw-tot-stream

- with start-all.sh script: just check that on sda/scripts/module tw-tot-stream key is uncommented.
- Using consumer-tw-tot/start-tw-tot-stream.sh:
  Provide the following options to the script or set the corrisponding environment variables:

SCRIPT ARGUMENT	ENV VARIABLE	DESCRIPTION
sda-home	SDA_HOME	The path of social-data-aggregator folder
with-master	MASTER	master name (eg local,spark://xxx.xxx)
spark-home	SPARK_HOME	The path to spark folder

BATCH DEPLOY:

In order to run consumer-tw-tot batch analytics start the shell script under the folder sda/scripts/consumer-tw-tot/start-tw-tot-batch.sh after providing the following settings:

- on sda/scripts/consumer-tw-tot/consumer-tw-tot-confs.cfg set the following properties:

KEY NAME	DESCRIPTION
MASTER	Spark master address (spark://MASTER_IP:MASTER_PORT) or local
SPARK_HOME	absolute path to spark home
INPUT_DATA_PATH	Default input data path (where raw data, on which analysis have to be done, are stored)

- **start-tw-tot-batch.sh** script:

```
./start-tw-tot-batch.sh --help
```

OPTIONS:

OPTION NAME	DESCRIPTION
from	time from which you want to start the analysis (ISO8601 format) e.g 2015-02-18T17:00:00+01
to	time to which you want to stop the analysis (ISO8601 format) e.g 2015-02-28T17:00:00+01
roundMode	define the round mode on the creation time. Possible options are: **min:**round on minute. **hour:**round on hour. **day:**round on day
granMin	**valid only if round mode is min.** Granularity,if you want to group in minute intervals (e.g gran=5 will group by 5 minutes -> the number of tweets in 5 minutes)

Consumer GRA (Gender Recognition Algorithm)

Introduction

On Twitter the information about user gender is not specified. Nonetheless, it is interesting having such an information for analytics purposes (e.g. for marketing research or having a clue if a target event was more interesting for male or for female users could be very useful). Providing support to business analytics is the reason why of our work: the development of a gender recognition algorithm (GRA) whose purpose is to classify the gender of twitter users.

For information on how the algorithm works and results achieved check the document on **/documents/GenderRecognitionAlgorithmGRA.pdf**

This consumer provides the information about gender on the aggregate information about tweet count (e.g 10 tweets made by males, 2 retweet made by females and so on).

There are two versions of this component:

- Stream
- Batch

Both modules are based on a core module which aim is to classify the gender of a twitter user from his profile information. The Gender Recognition Algorithm contains 3 sub algorithms:

name/screenName recognition

This sub algorithm expects key/value pairs in the form of name/gender. In its default implementation the module loads a file in the confs/consumers/consumer-gra folder called **names_gender.txt**. This file contains the key/value pairs in the following format:

```
name,gender
```

using comma as field separator. There are already some keys with the related gender.

The user can change the default implementation by implementing the interface NamesGenderMap. Then in GraConsumer.properties the property namesGenderMapImplClass has to be valorized with the qualified name of the new implementation. If the new implementation need some properties (for example db connection url) these can be added into the file *names_gender_mapping_impl.conf* in the form of key/value pairs.

recognize gender from profile description and colors

These two sub modules use internally a classifier. The classifier class must implement the MlModel interface providing an initialization method to train the classifier and a predict method to classify the gender of the target user providing a sparse vector of features. GRA core provides an implementation of MlModel with Naive Bayes with the class NBModel. The user anyway is free to change this implementation with a custom one implementing a different classifier. You can link the new implementation by edit *coloursModelImplClass* and *descrModelImplClass* properties in **GraConsumer.properties** file.

recognize gender from profile description

Create the training set and save it in LIBSVM format

Create a file containing training data with the following format:

```
<gender>FS<user profile description>

e.g.
m,the pen is on the table
```

where FS is the field separator. Then run the python script (located in $SDA_HOME/sda-tools/python_-scripts/sda_gra_tools/gra_usr_descr.py) that convert the training set in libsvm format (that will be used afterwards to feed the description gender classifier of GRA core):

```
$SPARK_HOME/bin/spark-submit --master local[*] gra_usr_descr.py --i <training data location> --a
    lgo tf
```

Where the algo option can be *tf* for term frequency algorithm or *tf-idf* for term frequency–inverse document frequency. **Remember to use *tf* algorithm to use this file for training in gra core even if you decide to apply tf-idf algorithm since the tfidf occurrencies will be calculated from gra description module. Use *tf-idf* in that case could lead to erroneous predictions.**

Below an example of the output file in libsvm format:

```
0 14955:1 16284:1 61154:1 86485:1 108074:1 168298:1 224032:1 228823:1 238246:1
0 228:1 6293:1 31852:1 66186:1 103560:1 109452:1 116014:1 132917:1 177241:1 194778:1 200529:1 22
    2879:1
0 50892:1 57911:1 140459:1 143926:1 198102:1 226265:1 246321:1 256253:1
1 84172:1 101480:1 168384:1 212544:1 252792:1
1 2091:1 33157:1 35412:1 39705:1 57535:1 70700:1 76150:1 92249:1 96011:1 104809:1 124240:1 12706
    1:1 207234:1 249431:3
```

recognize gender from profile color

Create the training set and save it in LIBSVM format

Create a file containing training data with the following format:

```
<gender>FS<profileBackgroundColor>FS<profileTextColor>FS<profileLinkColor>FS<profileSidebarFillColor>FS<profileSidebarBorderColor>

e.g.

m,9AE4E8,030202,0D0808,949B84,949B84
```

where FS is the field separator(, in the example). Then run the python script (located in $SDA_HOME/sda-tools/python_scripts/sda_gra_tools/gra_usr_color.py) that convert the training set in libsvm format (that will be used afterwards to feed the color gender classifier of GRA core):

```
$SPARK_HOME/bin/spark-submit --master local[*] gra_usr_color.py --i <training data location> --numcols 4 --nbits 9 --fdc
```

where:

- *numcols* is the number of profile colors to consider (over the 5 profile colors)
- *nbits* is the number of bits to which each color has to be scaled (for example from 24 to 9 bits in total -> 3 bits for each channel RGB)
- *fdc* (filter default colors): set this option if you want to filter twitter default colors configuration from the training set

Below an example of the output file in libsvm format (4 colors and 9 bits mapping):

```
0 1:1 8:1 234:1 445:1
0 1:1 445:1 481:1 512:1
0 1:2 8:1 445:1
0 148:1 284:1 365:1 373:1
0 74:1 154:1 303:1 375:1
0 1:1 74:1 102:1 311:1
0 1:1 66:1 147:1 302:1
```

GRA properties configurations

Property	Optional	Default	Description
coloursModelImplClass	YES	com.tilab.ca.sda.gra_core.ml.NBModel	class that implements the classificator for predictions from profile colours (Default implementation uses Naive Bayes classifier)
colorAlgoReductionNumBits	YES	9	number of bits to which scale each profile color (from 24 original bits). It determines the number of features in input for color classification algorithm
colorAlgoNumColorsToConsider	YES	4	The number of profile colors to consider (5 means all colors,1 just profile background color)
descrModelImplClass	YES	com.tilab.ca.sda.gra_core.ml.NBModel	class that implements the classificator for predictions from profile description (Default implementation uses Naive Bayes classifier)

Property	Optional	Default	Description
featureExtractionClassImpl	YES	com.tilab.ca.sda.gra_core.ml.FeaturesExtractionTFIDF	class that implements the feature extraction module. Two implementation are available: *FeaturesExtractionTF*,that implements Term frequency algorithm, and *FeaturesExtractionTFIDF* (Read https://en.wikipedia.org/wiki/Tf–idf for more information)
namesGenderMapImplClass	YES	com.tilab.ca.sda.gra_core.components.NamesGenderMapDefaultImpl	class that map keywords (person name or keywords to recognize pages e.g news) to gender (Default implementation is an in-memory hash map name/gender). Data for default implementation are stored under GRA configuration folder
trainingFilesPath	NO	x	Path where are stored GRA training files to feed classifiers (colors and descr). Use a distributed filesystem path to avoid undesidered behaviours

SETTING UP CONSUMER GRA

OVERVIEW

The consumer gra provides a per gender count on tweets, retweets, reply on geo and hashtags based criteria for a user defined time interval. There are two versions of this module:

- Stream
- Batch

Configuration

The confs/consumers/consumer-gra folder contains the following files:

dao_impl.conf

A properties file with the properties needed from the GraConsumerDao implementation. If you use the **GraConsumerDaoFileImpl** you need to provide a path on which save the data by editing the property *graOutputFilesPath*. If you prefere to save data on db there is a built in class implementation to save on dbms called *GraConsumerDaoHibImpl*. To switch to this implementation you need to edit the property *daoImplClass* on *GraConsumer.properties* file. In this case the *dao_impl.conf* file can be left blank but some configurations are needed on

gender-consumer-tw.cfg.xml

This file contains the configurations (connection url,username,password...) needed from *GraConsumerDaoHibImpl* to work properly.

```
<property name="connection.url">jdbc:mysql://localhost/twstats</property>

        <property name="connection.username"></property>

        <property name="connection.password"></property>



        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>

        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="current_session_context_class">thread</property>

        <property name="transaction.factory_class">org.hibernate.transaction.JDBCTransactionFact
            ory</property>
```

You can find the sql code to create the consumer-gra tables needed to store analytics result in social-data-aggregator/data_model in the project directory.

log4j.properties

the properties for log4j. Set where you want the connector log. Edit this file following your needs.

bus_impl.conf

This is the configuration file for the internal bus. By default is filled with apache Kafka configurations. If you want to use a different implementation please follow these steps:

1) Create a Java class that implements the BusConnection interface
2) Set the properties you need for your implementation into the bus_impl.conf file
3) Put the path to your implementation as the value for the property busConnImplClass into the **GraConsumer.properties** file (e.g "com.mypackage.MyImplClass")

GraConsumer.properties:

Common configurations

Key Name	Optional	Description
roundPos	NO	Decimal position on which round the latitude and longitude provided in case of geoLoc tweet (i.e. roundPos=3 , latitude=17.87654 -> latitude=17.876)
daoImplClass	NO	Java class that implements the ConsumerTwTotDao interface for the connection to the storage (default value: com.tilab.ca.sda.gra_consumer_dao.GraConsumerDaoFileImpl)

GRA Configurations

For GRA configurations please follow the guidelines provided on wiki page **Consumer GRA**.

STREAM CONFIGURATIONS:

Key Name	Optional	Description
keyRaw	NO	topic key for statuses raw (tweets as sent from twitter)
defaultRoundMode	NO	On which time field round to group values (allowed values for this property are: min,hour,day)
granMin	NO	Granularity in minutes (has to be the equivalent in minutes of the window duration)
numMaxCore	NO	Number of cores to associate to this application (in case you have to run multiple streaming application)
checkpointDir	NO	Directory where spark will save this application checkpoints
sparkBatchDuration Millis	NO	Duration of the batch (in milliseconds). It is the basic interval at which the system with receive the data in batches
sparkCleanTTL	NO	Duration (seconds) of how long Spark will remember any metadata (stages generated, tasks generated, etc.). Periodic cleanups will ensure that metadata older than this duration will be forgotten.

Key Name	Optional	Description
twTotWindowDura tionMillis	NO	Duration of the window. Both the window duration and the slide duration must be multiples of the batch interval. Data window on which analysis will be made.
twTotWindowSliding IntervalMillis	NO	Window sliding interval. The interval at which the window will slide or move forward. (set equal to the twTotWindowDurationMillis to avoid unexpected behaviour)
busConnImplClass	NO	Java class that implements the BusConnection interface for the interconnection with the internal stream bus (default: com.tilab.ca.sda.consumer.utils.stream.BusConnectionKafka Impl)

Consumer GRA deploy

STREAM DEPLOY:

To deploy consumer-gra-stream

- with start-all.sh script: just check that on sda/scripts/module gra key is uncommented.
- Using gra/start-gra-stream.sh: Provide the following options to the script or set the corrisponding environment variables:

SCRIPT ARGUMENT	ENV VARIABLE	DESCRIPTION
sda-home	SDA_HOME	The path of social-data-aggregator folder
with-master	MASTER	master name (eg local,spark://xxx.xxx)
spark-home	SPARK_HOME	The path to spark folder

BATCH DEPLOY:

In order to run consumer-gra batch analytics start the shell script under the folder sda/scripts/consumer-gra/start-gra-batch.sh after providing the following settings:

1) on sda/scripts/consumer-gra/consumer-gra-confs.cfg set the following properties:

KEY NAME	DESCRIPTION
MASTER	Spark master address (spark://MASTER_IP:MASTER_PORT) or local
SPARK_HOME	absolute path to spark home
INPUT_DATA_PATH	Default input data path (where raw data, on which analysis have to be done, are stored)

1) **start-gra-batch.sh** script:

```
./start-gra-batch.sh--help
```

OPTIONS:

Option Name	Descripion
from	time from which you want to start the analysis (ISO8601 format) e.g 2015-02-18T17:00:00+01

Option Name	Descripion
to	time to which you want to stop the analysis (ISO8601 format) e.g 2015-02-28T17:00:00+01
roundMode	define the round mode on the creation time. Possible options are: **min:**round on minute.**hour:**round on hour.**day:**round on day
granMin	**valid only if round mode is min.** Granularity,if you want to group in minute intervals (e.g gran=5 will group by 5 minutes -> the number of tweets in 5 minutes)
I	Override the default input data path (the source where to read input data