INTEGRATION OF BIG DATA INFRASTRUCTURE ANALYSIS AND VISUALIZATION OF GDELT AND OPEN MEASURES DATA

EXAMINING INFORMATION SYSTEM FOR BIG DATA

Giuseppe Bello Luigi Mascolo Alfonso Marino

Summary

1 Introduction

1.1 Purpose of the project

The project aims to study how GDELT's most relevant news influences activity on social media, particularly on platforms such as TikTok, Truth, and VK. The social data will be extracted through the Open Measures API, which allows for a whole range of information on posts/videos/comments published online, while the GDELT data are obtained by directly leveraging daily csv files made available by the organization. In addition to a general analysis for GDELT news and content extracted from Open Measures, an attempt is made to analyze what is happening in the world by looking at what are the top news stories, and how they impact social media.

To handle the huge volume of data extracted from GDELT and Open Measures, the project uses Hadoop to distribute the data across multiple nodes and Apache Spark to process this data in a parallel and scalable manner, ensuring that the workload is distributed efficiently across multiple computational resources.

The collected data is then analyzed and visualized through Grafana, which enables the creation of interactive dashboards to monitor trends in social metrics in relation to global events. Data storage takes place on MongoDB, while communications between MongoDB and Grafana are managed through the MongoDB API.

1.2 Technologies Used

Docker: A tool used to manage containers and ensure an isolated and easily replicable environment for project execution.

Python: Main programming language used to collect and analyze data from GDELT, extract keywords, make requests to the Open Measures API to collect data from socials, and manage the entire workflow.

Hadoop: Used for managing and processing large volumes of data. Hadoop distributes data across multiple nodes, enabling parallel processing and scalable management of data extracted from GDELT and Open Measures.

Apache Spark: Used to process and analyze data quickly and efficiently. Spark enables parallel processing of data on multiple nodes, greatly improving performance in processing large amounts of data. It is used for keyword analysis, extracting insights, and processing data from GDELT and Open Measures.

MongoDB: NoSQL database used to store data collected from GDELT, social media, and analyzed information. MongoDB offers scalable and flexible storage.

Grafana: Data visualization platform used to create interactive dashboards, which allow users to explore and monitor the impact of global news on social media dynamically.

MongoDB API: The MongoDB API is used to connect the database to Grafana, allowing the collected and stored data to be viewed in real time.

2 Use

2.1 Structure

The project is divided into two main folders.(Figure 1)

```
Project/
├── app/
│   ├── aggregations/
│   │   ├── __init__.py
│   │   └── aggregation_collections.py
│   ├── downloaders/
│   │   ├── __init__.py
│   │   ├── gdelt_downloader.py
│   │   └── openm_downloader.py
│   ├── grafana-provisioning/
│   │   ├── datasources/
│   │   │   └── datasource.yml
│   │   └── dashboards/
│   │       ├── dashboard.yml
│   │       └── gdelt_dashboard.json
│   ├── processors/
│   │   ├── __init__.py
│   │   ├── gdelt_spark_processor.py
│   │   └── openm_spark_processor.py
│   ├── utils/
│   │   ├── __init__.py
│   │   ├── keyword_extractor.py
│   │   ├── schema_registry.py
│   │   └── spark_singleton.py
│   └── main.py
├── mongodb-api/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── endpoint.py
├── docker-compose.yml
├── Dockerfile
└── requirements.txt
```

Figure 1

The app folder in turn contains:

- Aggregations folder, for the script for creating the aggregated collections;

- Donwloader folder, containing python scripts for downloading data from GDELT and Open Measures;
- Graphana-provisioning folder that in turn contains two other folders called dashboards and datasources that instead contain json and yml files for creating the explorable dashboard in Grafana;
- Processor folder, which contains the scripts for processing the data;
- utils folder, containing scripts used in other operations (e.g., for keyword extraction and schema creation for Open Measures data, and spark connection management).
- Script main

The second main folder called mongo-api contains:

- Python script called endpoint.py with which you go to set the settings for the connection of databases in MongoDB with Grafana, and where endpoints are created.
- Dockerfile that contains the docker container settings;
- Requirements.txt file that includes the libraries needed for proper execution of python scripts.

We then have the docker-compose.yml and dockerfile files that include all the information needed for the creation and proper operation of the containers, respectively. Finally we again find the requirements.txt file containing other libraries used in the python scripts in the app folder.

2.2 Quick Guide

2.2.1 Prerequisites

It is sufficient to have an updated version of Docker Desktop for the project to work properly. If not, it is necessary to upgrade the software or perform a new installation. (https://www.docker.com/products/docker-desktop/)

2.2.2 Startup example

After starting Docker Desktop, locate in the ''project folder,'' open the command prompt (pointing to the 'project folder'') and run the following command: *docker compose up -d*

This command will proceed to start the creation of the containers. When the process is complete, you will be able to monitor the containers directly in Docker Desktop. It is suggested to open the container "g-delt processor" and observe the logs to check for errors during the process. The execution will be terminated when the container g-delt processor "shuts down."

2.2.3 Opening the dashboard

To view the final interactive dashboard, simply go to the following host localhost:3001, enter admin as your username and password, and log in. The final dashboard can be viewed in the "Dashboards" section of Grafana.

3 Python Script Description

Figure 2 briefly explains the relationships between the scripts, and l3 structures supporting the various operations that are performed, which will be discussed in more detail later. The proposed diagram is to be interpreted as a guideline for understanding where all the classes and functions used in the main script are derived from.

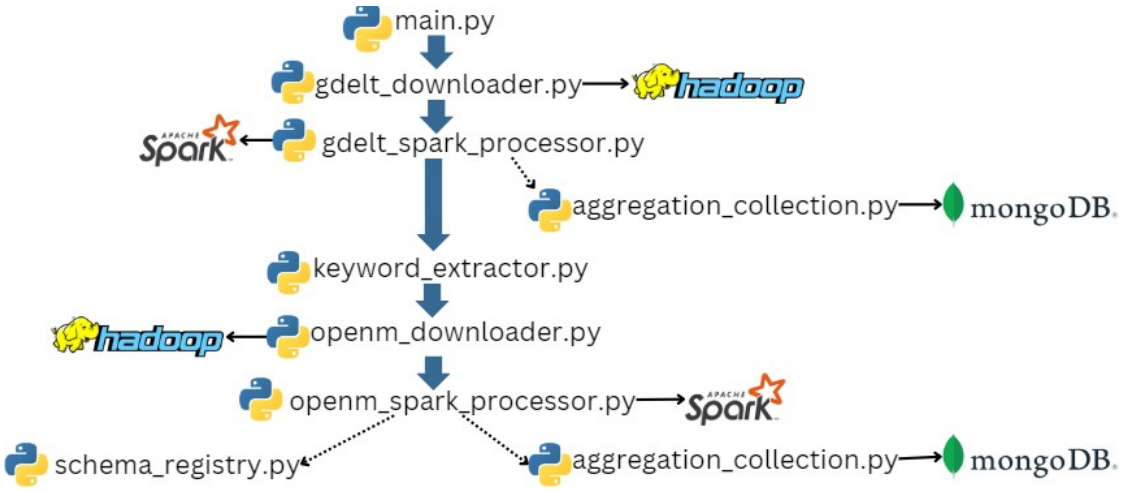

Figure 2

3.1 main.py

The main.py code is a script primarily designed to download, process, and aggregate data.

Initial setup: A time interval of 30 days is calculated and the connection is established to the HDFS the environment variables HDFS_HOST and HDFS_PORT.

Module initialization: Instances are created for GDELT and OpenMeasures downloaders and processors, whose code will be analyzed later.

Downloading and processing GDELT data: GDELT files related to established time interval are downloaded using GDELTDownloader. Each file is processed using SparkProcessorGDELT. Global aggregations are performed on the processed data.

Aggregations and keyword management: Keywords are extracted from the main GDELT news items, via AggregationsManager and the keyword_extractor module.

Downloading and processing OpenMeasures data: Data from specific social media, defined in openmeasures_social (TikTok, Truth Social, Vk), are downloaded. The downloaded files are processed using SparkProcessorOpenMeasures, with subsequent aggregation steps.

Error handling: The script is structured with try/except blocks to handle any errors download and processing, while still continuing to process other data.

Resource closure: Finally, we close the Spark session via SparkManager.stop() to free up resources.

3.2 gdelt_downloader.py

The gdelt_downloader.py code defines the GDELTDownloader class, which handles downloading files from GDELT and uploading them to HDFS.

Initialization (__ init__): Sets the URL of HDFS (default: http://namenode:9870). Defines the maximum number of connection attempts to HDFS (max_retries). Initializes the HDFS client via _init_hdfs_client.

Connection to HDFS (_init_hdfs_client): Tries to connect to HDFS using the InsecureClient module of hdfs. Make several attempts (up to max_retries) with a 10-second pause between each attempt. If the connection fails after all attempts, it raises an exception.

Downloading a single file (download_gdelt_file): Creates the URL to download a GDELT file related to a specific date. Downloads the ZIP file from the URL. Unzips the ZIP file in memory and loads the CSV file into HDFS: Creates the necessary HDFS directory, if it does not already exist. Writes the contents of the CSV to the HDFS destination. Returns the HDFS path to the uploaded CSV file. If the download or upload fails, raises an exception.

Downloading a date range (download_date_range): Downloads GDELT files for all dates between start_date and end_date. For each date: Tries to download and upload the file to HDFS. If the download fails for a specific date, logs the error and continues with subsequent dates. Returns a list of HDFS paths to the downloaded files.

3.3 gdelt_spark_processor.py

The code defines a Python class, SparkProcessorGDELT, to process data from the GDELT dataset.

Initialization (init): Sets up logging to monitor execution. Retrieves environment variables (hosts and ports for HDFS and MongoDB). Initializes an HDFS client to read files from HDFS. Gets a shared instance of Spark via the SparkManager class. Configures a MongoDB client to store the processed data.

Process_file method: Loads a CSV file from HDFS using a predefined schema. Counts the number of raw records and applies transformations with _transform_data. Converts processed data into dictionaries and writes them to MongoDB in blocks of 100,000 records to improve efficiency. Records statistics (records processed, saved, etc.).

Data Schema (_get_schema): Defines the columns in the CSV file with their respective data types (e.g., StringType, IntegerType).

Data transformations (_transform_data): Adds columns with timestamp and processing date.

Aggregations (process_all_aggregations): Uses the AggregationsManager class, (realized in the aggregation_collection.py scrpit) to calculate statistics and aggregations, such as daily analysis, trends, and top mentions. Extracts keywords from the processed dataset using the KeywordExtractor class (realized instead in the scrpit keywird_tractor.py)

Cleanup (close): Closes connections to MongoDB and Spark to free up resources.

3.4 keyword_extractor.py

This script implements a Python class, KeywordExtractor, designed to extract keywords from a URL, using the spaCy library for natural language processing.

Initialization (__ init__): Loads the spaCy language model (en_core_web_sm). Configures a list of stop words, i.e., common words to be excluded. Initializes a logger to monitor for errors.

Title segment extraction (_extract_title_segment): Analyzes the path (path) of the URL to identify the most relevant segment, i.e., the "title." Excludes technical segments such as UUIDs, hashes, API paths, etc. Cleans segments of symbols and file extensions. Uses spaCy to identify the segment with the most significant content, such as nouns and verbs, based on a calculated score.

Keyword extraction (extract_keywords): Analyzes the title segment extracted from the URL. Identifies significant keywords, such as nouns, verbs, adjectives, and acronyms. Uses word lemma (base form) to avoid duplicates due to grammatical variations. Limit keywords to max_keywords.

3.5 openm_downloader.py

The script openm_downloader.py is designed to download data an Open Measures API, save it in JSON format, and upload it to an HDFS system.

HDFS Client(init) initialization: Connects the application to HDFS using the provided URL (hdfs_url). Retries the connection up to max_retries times in case of failure.

Downloading JSON(download_openmasures_file) files: Downloads data about a social media platform specified in the main script, for a date range also specified in the main. Makes an HTTP request to the Open Measures API using a JWT token (JSON Web Token) for authentication. Saves the downloaded JSON file in HDFS.

Managing date ranges(download_date_range): Download data in three-day blocks to avoid large queries. Builds the "search term" from the keywords to get the results.

Messages and Logs: Prints information about the status of the HDFS connection, date periods processed, and the total number of documents found. Handles exceptions and continues the process for subsequent periods in case of errors.

Path return: Returns a list of paths to JSON files uploaded to HDFS.

3.6 openm_spark_processor.py

With this script is implemented, the SparkProcessorOpenMeasures class, to process data obtained from Open Measures.

Initialization(init): Sets up logging to monitor the process. Reads the environment variables needed to connect to HDFS and MongoDB. Configures a shared instance of Spark via a singleton class (SparkManager which will be seen later). Establishes an HDFS client and a MongoDB instance for the openmeasures_db database.

File processing(process_file): The process_file function reads a JSON file from HDFS and transforms it using a platform-specific schema (TikTok , Truth Social or Vk). The data is

loaded into a Spark DataFrame, transformed and normalized according to the platform. The results are counted and then saved in MongoDB in batch to reduce the load.

Data transformations(_transform_data): The _transform_data function applies platform-specific transformations: such as extracting hashtags, converting timestamps, and normalizing fields such as post id, likes, views. Finally, it calculates an additional field called total_engagement that sums likes, shares, and comments.

Aggregations(process_all_aggregations_openm): The process_all_aggregations_openm function creates aggregation collections based on the processed data, such as temporal analysis, platform content, influencer analysis, and integration with GDELT.

Resource management: The close method ensures that connections to Spark and MongoDB are securely closed.

3.7 schema_registry.py

This code defines a class called SchemaRegistry, which serves as a schema registry for managing data from the different social platforms under consideration.

Defining specific schemas for each social media outlet: Each platform, such as TikTok, Truth Social, and VK, has a detailed schema for representing data about videos, posts, statistics, and other information. These schemas are defined using StructType and StructField from PySpark.

Management via a registry: The SCHEMA_REGISTRY mapping associates the names of social platforms, such as "truth_social" , "tiktok_video "and "vk," to functions that return their respective schemas.

Static method to get schema(get_schema): The get_schema method accepts the name of a social platform and returns the corresponding schema. If the platform is not supported, an exception is raised.

Modular use: Each scheme is separate to facilitate maintenance and the addition of new social platforms in the future.

3.8 aggregation_collection.py

The code implements a class that goes to handle the creation of different aggregations that will be used to create the final results dashboard.

Configuration and Connection: A MongoDB client is initialized with the gdelt_db and openmeasures_db databases.

Main Methods:

- create_daily_insights: Generates a collection called daily_insights by calculating daily metrics such as average impact, average tone, and number of mentions from existing events. Sort the data by date and create an index on the date to optimize queries;
- create_geo_analysis: Creates a geo_analysis collection with statistics based on geographic coordinates (latitude and longitude). Filters valid coordinates, groups events and considers only those with at least 1000 mentions. Creates indexes to optimize geographic searches;

- create_trend_analysis: Generates a trend_analysis collection with temporal trends based on the total number of mentions per day. Sorts the data by date and creates an index for temporal queries;
- create_top_mentions: Creates a top_mentions collection by selecting the 10 events with the most total mentions. Adds keywords extracted from URLs using a keyword_extractor;
- get_top_mentions_keywords: Extracts the 20 most cited URLs (Top Mentions) from the events collection by calculating a total based on mentions, sources, and articles. For each of the URLs, keywords are extracted using keyword_extractor. Next, the keywords are counted and sorted by frequency, returning a dictionary of the sorted keywords;
- create_social_temporal_metrics: Aggregates daily time metrics from social media data, such as number of posts, interactions (engagement), likes, shares and comments. The function also calculates the percentage of sponsored content and stores the results in the social_temporal_metrics collection;
- create_top_authors_engagement: Determines authors with the highest total engagement. Sorts the results by decreasing engagement, selecting the top 10 authors and storing the data in the top_authors_engagement collection;
- create_gdelt_social_correlation: Compares daily GDELT data (number of mentions) with social media data (total engagement). It combines this information by date and stores the results in the gdelt_social_correlation collection;
- create_all_collections: Coordinates the creation of all previously defined collections and collects keywords from Top Mentions. Logs the results and returns the overall success of the operations and the extracted keywords.

Close method: Closes the connection to the MongoDB database to free up resources.

3.6 spark_singleton.py

This code defines a SparkManager class that implements a singleton to manage SparkSessions.

Singleton for SparkSession: The class ensures that there is only one instance of SparkSession during the entire program execution. If the instance is not present, it is created and configured; otherwise, the existing instance is returned.

SparkSession configuration: configure Spark with various settings: Local use: master("local[*]") allows all available cores to be used on the local machine. Driver memory: spark.driver.memory allocates 4 GB of memory to the Spark driver. Partitions: spark.sql.shuffle.partitions and spark.default.parallelism define the default number of partitions for shuffle and parallelism operations. Timeout and heartbeat: spark.network.timeout and spark.executor.heartbeatInterval ensure that Spark handles connections and timeouts correctly. HDFS: Configures the use HDFS, specifying the host (hdfs_host) and other settings such as data replication.

Stop method: Stops the SparkSession instance if it exists and sets it to None, allowing resources to be freed.

3.10 endpoint.py

The code, located in the mongodb-api folder, defines a Sanic-based REST API that interacts with MongoDB databases to provide data to a Grafana interface.

Initial configuration: Import the necessary libraries, including Sanic, Motor (for MongoDB), and logging, and configure the MongoDB client to connect to the openmeasures_db and gdelt_db databases.

Function mongo_to_dict: Converts MongoDB objects (e.g., ObjectId, datetime, and NaN values) to JSON-compatible formats.

The API defines several endpoints for Grafana, each querying MongoDB databases to provide pre-calculated data, geographic analysis, temporal analytic trends, social time metrics, top authors sorted by engagement, analysis of engagement distribution by platform, and correlation data between GDELT and social metrics.

Each endpoint includes a try-except block to handle exceptions and log errors.

4. Description docker-compose.yml

In summary, the file configures a distributed architecture to process, store and visualize data, using Hadoop, Spark, MongoDB and Grafana, with each component communicating via the Docker network.

Specifically, the following services are configured :

- namenode: Starts a container with Hadoop NameNode, which manages file system metadata HDFS. It exposes ports 9870 and 9000 for registration and communication with DataNodes.
- datanode: Starts a container with Hadoop DataNode, responsible for storing data. Connects to the NameNode via HDFS.
- spark-master: Starts the Apache Spark master, which coordinates Spark workers. Exposes ports 8080 (web interface) and 7077 (Spark communication).
- spark-worker: Starts an Apache Spark worker that executes distributed tasks. Connects to the Spark master using the specified URL.
- mongodb: Starts a MongoDB container, with the database accessible on port 27018, used to store analysis data.
- mongodb-api: Creates an API application (probably REST) that connects to MongoDB, exposing the API on port 3000.
- grafana: Starts an instance of Grafana to display data, configuring dashboard provisioning. Depends on mongodb-api for data access.
- gdelt-processor: A custom application that interfaces with Hadoop, Spark and MongoDB to process GDELT and Open Measures data, using Spark distributed processing and MongoDB for saving results.

5. Description of GDELT and Open Measures fields.

5.1 GDELT

GDELT is a huge database that tracks global, geopolitical, and media events in real time, gathering information from news sources around the world. Its data are organized into structured fields, each of which offers specific details about events, actors, positions, and sentiments. Here is a more in-depth description of the most important fields:

1. ID and Date:

- GlobalEventID: Unique identifier of each event. It is used to refer precisely to a specific event in the database.
- Day: The date of the event, represented in the format YYYYMMDD. It is useful for temporal analysis or
  To filter events by period.

2. Actors Involved:

- Actor1Name and Actor2Name: The names of the two main actors involved in the event, which can
  Represent individuals, governments, organizations or social groups.
- Actor1CountryCode and Actor2CountryCode: Country codes (ISO-3166) associated with the two actors. This field is useful for tracking events involving specific countries.

3. Encoding of Events:

- EventCode: A standardized code derived from the Conflict and Mediation Event Observations (CAMEO) system that classifies the type of event (e.g., protests, attacks, negotiations). It allows systematic analysis of event categories.
- EventRootCode: The main category of the event (the "root" of EventCode). For example, a code "01" represents general diplomatic interactions, while "14" refers to material violence.
- QuadClass: Rank events in one of four major categories:

    1. Verbal Cooperation
    2. Material Cooperation.
    3. Verbal Conflict.
    4. Material Conflict.

4. Geolocation:

- ActionGeo_FullName: Full name of the location where the event occurred.
- ActionGeo_Lat and ActionGeo_Long: The geographical coordinates (latitude and longitude) of the
  location of the event, enabling visualization on maps or geographic analysis.
- ActionGeo_CountryCode: The ISO country code associated with the location.

5. Context and Impact:

- NumMentions: Total number of times the event was mentioned in articles or news stories. Indicates the media relevance of the event.
- NumSources: Number of unique sources reporting the event, useful for estimating the reliability or
  information diversity.

- GoldsteinScale: A scale from -10 to +10 that measures the geopolitical impact of the event, where values
  positive values represent cooperative events and negative values represent conflicting events.
- AvgTone: The average tone of articles about the event, with values ranging from -100 (very negative tone) to +100 (very positive tone). Used for global sentiment analysis.

5.2 Open Measures

The OpenMeasures API enables the extraction of structured data from social networks such as TikTok, Truth Social, and VK. This data includes information on interactions, content posted, and users, useful for engagement analysis, information dissemination, and sentiment. Here is a description of the main fields for each social:

1. TikTok Video:

TikTok provides data on video content, often related to trends, hashtags and interactions. Key fields include:

Video data:

- video_id: Unique identifier of the video.
- description: Description or text associated with the video (often includes hashtags or trending tags).
- duration: Duration of the video (in seconds).
- hashtags: List of hashtags used in the video, useful for trend analysis.
- sound_title: Title of the audio or soundtrack used in the video.

Engagement metrics:

- like_countNumber of likes received by the video.
- comment_count: Number of comments.
- share_count: Number of times the video was shared.
- view_count: Total number of views.

About the author

- user_id: Unique identifier of the content creator.
- username: Username of the author.
- followers_count: Number of followers of the author (at the time of data collection).
- verified: Boolean indicating whether the account is verified.

2. Truth Social

Truth Social is a platform often associated with news and opinion sharing, with a similar structure to Twitter. Main fields include:

Post data:

- post_id: Unique identifier of the post.
- content: Text of the post, often containing opinions, statements or external links.
- timestamp: Date and time the post was published.
- media_urls: URLs of attached media (images, videos or documents).

Engagement metrics:

- like_count: Number of likes received.
- reply_count: Number of replies or comments to the post.
- retruth_count: Number of "retruths" (similar to

retweets). Author information

- user_id: Unique identifier of the author of the post.
- username: Username of the author.
- followers_count: Number of followers.
- verified: Boolean indicating whether the account is verified.

3. VK

VK (VKontakte) is the leading social network in Russia, with similar functionality to Facebook. Extracted data concern posts, interactions, and user information.

Post data:

- post_id: Unique identifier of the post.
- content: Text of the post, which may include opinions, links or media.
- timestamp: Date and time of publication.
- media_attachments: List of attachments, such as images, videos or

documents. Engagement metrics:

- like_count: Number of likes received.
- comment_count: Number of comments.
- share_count: Number of shares.
- view_count: Number views.

Author information

- user_id: Identifier of the user who published the post.
- username: Name of user (or group if it is a post on a page).
- followers_count: Number of followers of the user or group.

6. Useful Links

GDELT: https://www.gdeltproject.org

Open Measures : https://openmeasures.io

Docker Desktop: https://www.docker.com/products/docker-desktop/

Apache Spark: https://spark.apache.org

Apache Hadoop : https://hadoop.apache.org

MongoDB : https://www.mongodb.com

Grafana : https://grafana.com