Big Data-driven Platform for Cross-Media Monitoring

Liana Napalkova, Pablo Aragón, and Juan Carlos Castro Robles

Eurecat Technology Centre
C/ de Bilbao, 72, 08005 Barcelona, Spain
Email: {liana.napalkova, pablo.aragon, juancarlos.castro}@eurecat.org

***Abstract*—The abundance of online media content requires highly scalable architectures to allow cross-media monitoring. This paper presents an innovative big data-as-a-service platform for analysing large complex networks in order to enhance cross-media monitoring. In contrast to the existing media monitoring systems, the platform equips marketers with several distinctive features. First, while most of the systems perform quantitative exploratory analysis of social media, our platform applies graph analytics in order to reveal social interaction types, hidden patterns in the cross-media network and the information diffusion over time. Second, our platform integrates and implements distributed versions of graph analytics algorithms (Louvain, HITS and others) that can scale to a large volume of data. Third, the creation of cross-media graphs is triggered by user-defined queries that can be easily specified by marketers. Thus, end-users can build and analyse different graphs according to specific goals of the study. Finally, the platform allows reducing Hadoop cluster usage costs due to executing the graph mining algorithms on demand triggered by user-defined queries. Instead of running costly streaming processes that continuously listen for new queries, we implemented Spark-as-a-service approach via Apache Livy REST interface.**

***Index Terms*—Cross-media, big data, distributed computing, graph mining, network analytics, large networks.**

I. Introduction

Online social networks have become fundamental spaces in modern societies for discussing content of any topic. Millions of users participate actively in social media to share their experiences as consumers and, also, to review opinions from other users on services and products of their interest. For this reason, many entities from the marketing sector have shown great interest in developing systems and applications in recent years in order to collect these valuable fingerprints enriched with other media data sources (for example, online news, blogs, forums, etc.) [1].

The possibilities offered by the abundance of social and other media content are numerous, for example, conducting market research, measuring the success of advertising campaigns, identifying influential actors who can spread messages, brand awareness, etc. On the one hand, having cross-media monitoring systems that automate these tasks is a great motivation as it can have a positive impact on the return on investment. On the other hand, this abundance of cross-media content becomes a technological challenge of great importance due to the need to develop highly scalable architectures [2].

While a large majority of platforms have typically focused on content analysis [3], [4], network analysis is often ignored. This gap is of great relevance because cross-media systems are typically described in terms of complex networks, for example, conversations in online social platforms usually have a topology of interconnected nodes combining organization and randomness. The typical size of large networks counts in millions of nodes and these scales demand new methods to retrieve comprehensive information from their structure. Retrieving information from the data and searching for meaning in that data is the key differentiating factor for competition.

II. Related Work

Nowadays marketers dispose of numerous monitoring platforms that are capable of discovering the large volume of data from social, blogs, forums, news and other data sources. Among such platforms one can mention Brandwatch[1], Hootsuite[2], Boardreader[3], etc. These platforms provide excellent toolkits for the media content analysis and they can answer the questions such as, for example, what types of contents motivate users to be engaged in a particular page or what is users attitude towards a particular topic or product, etc. In addition, most of these platforms support content publishing and moderation.

Typically, the existing media monitoring platforms provide dashboards that summarize quantitative analysis outputs in the form of pie charts, bar charts, line plots, etc. However, there is a lack of platforms capable of delivering a sophisticated qualitative analysis revealing social interaction types, hidden patterns in the network and the information diffusion over time. Therefore, improved metrics of social network analytics are highly demanded by marketers.

One of few platforms that performs qualitative analysis of the network is Audiense[4]. This tool evaluates audiences and influencers in order to help marketers make decisions and discover new opportunities in activating audiences across multiple online and offline channels. However, the solution lacks a dynamic analysis of content diffusion over time. Moreover, the analytical results are provided at a high level of

[1]https://www.brandwatch.com
[2]https://hootsuite.com
[3]http://boardreader.com
[4]https://audiense.com/

granularity, which is very helpful to get a global vision, but might be insufficient to retrieve more specific conclusions.

Thus, the progress and innovation are no longer hindered by the ability to just, collect and integrate data from various sources for further quantitative analysis. The major deal consists of extending toolkits aimed at discovering the opportunities delivered by these data sources. In particular, taking into account that cross-media networks have non-trivial topological features, efficient scalable graph mining algorithms should be used to enrich the conclusions of traditional statistical analysis.

A. *Challenges to be met*

To enable the marketers with new tools that help answering more sophisticated questions, for example, how a viral content diffuses in the network over time, several technical challenges should be solved:

- How to flexibly create new graphs that better fit the goals of particular marketing analysis?
- How to scale and optimize graph mining algorithms to handle large complex networks?
- How to efficiently persist and query the graphs representing cross-media networks that exceed the computation and memory capacities of a single processor?
- How to visualize large complex networks in an interpretable way?

Therefore, we present in this paper how a big data technological stack provides the power to analyze cross-media networks using techniques of network science at various levels of granularity, while offering solutions to the aforementioned challenges.

III. Overview of distributed graph persistence systems

In this section, we analyze one of the challenges stated in the previous section. In particular, we investigate the applicability of four different storage systems (Neo4j[5], JanusGraph[6], Elasticsearch[7], and OrientDB[8]) to the persistence of large complex networks. To compare the storage systems, the following features have been identified:

(a) *Graph storage format* specifies how graphs are persisted: (i) using a relational database or any other general-purpose data storage; (ii) using native graph storage that is optimized for graphs, ensuring that data are stored efficiently by writing nodes and relationships close to each other; (iii) using multi-model database that is organized around various data models.

(b) *License* defines the use and redistribution of software for different purposes (commercial and non-commercial).

(c) *Partitioning method* identifies how data are divided across multiple instances of the schema to meet the demands of data growth. As the size of the data increase, a single instance may not be sufficient to store the data nor provide an acceptable read and write throughput.

(d) *Graph query language* defines how the data are selected, inserted, updated or deleted from the graph.

[5]https://neo4j.com
[6]http://janusgraph.org
[7]https://www.elastic.co
[8]https://orientdb.com

Table I summarizes the analyzed graph persistence systems. Based on these criteria, the system presented in this paper is powered by OrientDB due to the following beneficial features:

- Multi-model database combines native graph storage (for example, Neo4j or JanusGraph) with search features (for example, Elasticsearch).
- It demonstrated a good performance in running successive local queries [5].
- The technology is free for comfmercial purposes.
- Graph queries are done using a well-known SQL language with some minor modifications, which makes the learning curve considerably shorter.
- Data sharding is supported at class level.
- Multi-tenancy is also supported, that is each user (tenant) of the platform will be able to access to a separate distributed graph database.

IV. Distributed Graph Analytics Platform

In this section we present the Distributed Graph Analytics Platform (DGAP) that is able performing qualitative and quantitative analysis of cross-media networks at a low level of granularity. The platform handles raw data that arrive from numerous content providers, such as social media, online news, blogs, forums, etc. This DGAP is a completely independent system that can be integrated into any existing technological stack (see Figure 1). The description of the platform is organized to address challenges mentioned in II.A.

A. *Overall data pipeline*

The DGAP treats a cross-media network as a graph. Graph nodes are authors, mass-media outlets, documents or any other entity. Graph edges represent interactions between the adjacent entities. In particular, our system supports the following types of interactions:

- *Interactions between documents published by authors*:
  - URL: a document contains a web link to another document.
  - Similar contents: citations and sharing of a partial content of a document.
  - Comments in a social network: Facebook comments to posts.
- *Interactions between a document and an entity*: a document cites an author or another entity, for example, Twitter mentions.
- *Interactions between an entity and a document*: an author interacts with a document, for example, Twitter favorites.
- *Interactions between entities*: static relation, for example, followship of users.

In order to build the resulting graph and to retrieve knowledge from it, the platform executes a workflow. This workflow is triggered by an input query as indicated in Figure 2. The

Table I: Graph persistence systems

Features	Elasticsearch	Neo4j	JanusGraph	OrientDB
Graph storage	Non-native graph storage	Native graph storage	Native graph storage	Native graph storage
License	Apache Version 2	GPL v3	Apache Version 2	Apache Version 2
Partitioning	Sharding	No partitioning of a graph	Depends on the used storage backend	Sharding
Graph query	Graph Explore API	Cypher for Spark	Gremlin	Supports SQL queries

query includes the rules that determine how a graph should be created based on raw data. The usage of such query gives a full control over the definition of a graph.

The major steps of the workflow are given below:

1) Receive the user-defined query through the message broker software RabbitMQ[9].
2) Parse the query and retrieve explicit, implicit and cluster rules for building a graph.
3) Submit the Spark job to Apache Livy server via REST interface.
4) Perform the following steps in Spark:
   - Construct the graph based on retrieving data from Elasticsearch according to explicit, implicit and cluster rules.
   - Apply the graph mining algorithms in order to extract communities and other knowledge from the data (roles, authorities, hubs, degrees).
   - Collect the properties of communities, users and contents.
   - Save communities, users and contents in a distributed database (OrientDB).
   - Notify the front-end (via API) about finishing the calculation process.

Next, we provide more details about the major steps of the workflow by commenting how each of these steps solves challenges in II.A.

B. How to flexibly create new graphs that better fit the goals of particular marketing analysis?

To provide a custom way for creating graphs, we introduced the concept of an input query. The query specifies how a graph should be created based on existing raw data that can be stored in Elasticsearch or any other data storage system. The query, encoded as a JSON string, arrives to the DGAP platform via messaging broker called RabbitMQ. The reason of using RabbitMQ broker is a complete decoupling of the query production and consumption services, which enables the communication between multiple independent systems, for example, the DGAP platform and the client system that uses this platform as a service.

[9]https://www.rabbitmq.com

The creation of a graph is based on three different rules: *explicit*, *cluster* and *implicit*. Each of these rules specifies how nodes and edges of the graph should be generated.

Explicit rule is used to create edges from the original node that created a document to the node that re-published this document.

Cluster rule establishes a relationship between the original document and all other documents that belong to the same category. The cluster field defines an attribute that should be common for all its entities.

Implicit rule defines a hidden relationship between entities. For example, in case of two tweets the edge is created when an original tweet includes a link to a publication that, according to the rule, becomes a target document. Another example is a relationship between documents that share the same domain name.

C. How to scale and optimize graph mining algorithms to handle large complex networks?

To scale and optimize graph mining algorithms aimed at building graphs according to the above-described rules, we used Apache Spark cluster-computing framework and Scala programming language. Every time the graph mining algorithms should be executed, a new HTTP request is submitted to

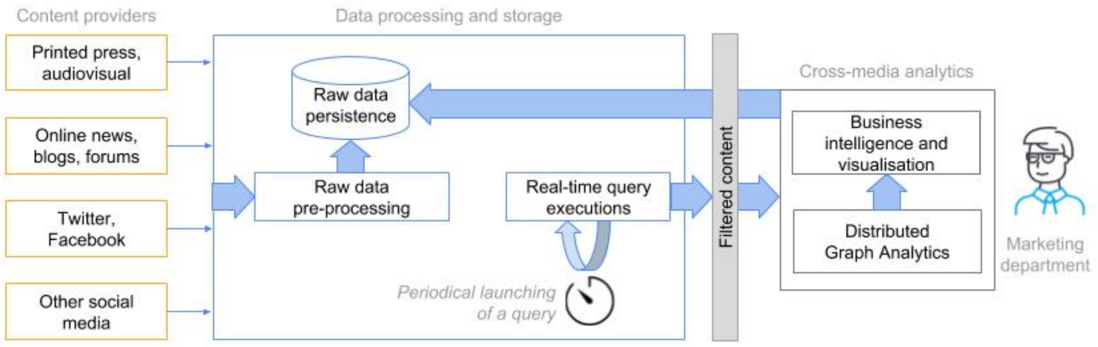

Content providers
Data processing and storage
Printed press, audiovisual
Online news, blogs, forums
Twitter, Facebook
Other social media
Raw data persistence
Raw data pre-processing
Real-time query executions
Periodical launching of a query
Filtered content
Cross-media analytics
Business intelligence and visualisation
Distributed Graph Analytics
Marketing department

Figure 1: Conceptual scheme of the Big Data-driven Platform for Cross-Media Monitoring.

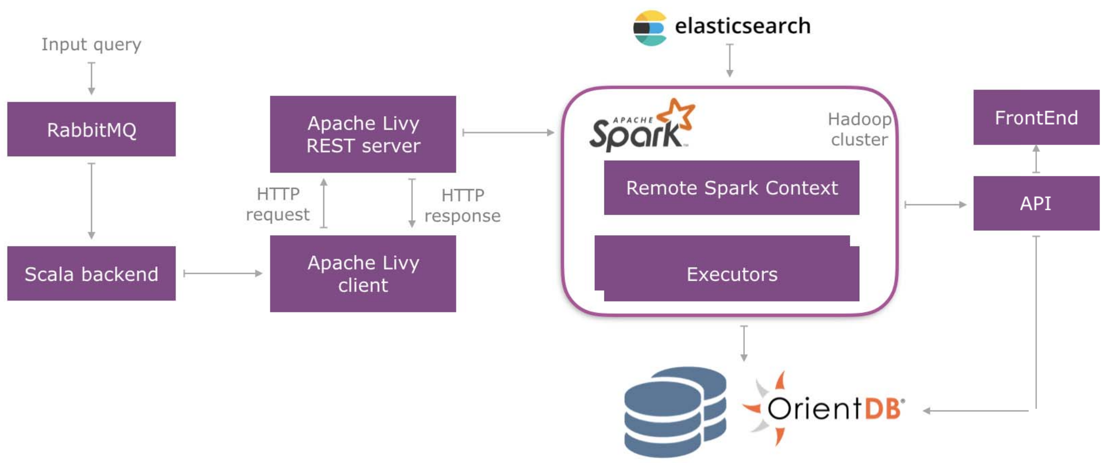


Figure 2: The DGAP architecture

the Apache Livy server that sends a batch Spark job to a cluster via REST interface. This approach considerably reduces the total cost of running Spark cluster, because there is no need to maintain a Spark streaming job that listens to RabbitMQ queue for new input queries. The main program that consumes queries from RabbitMQ is written in Scala, and the Spark application is launched on request only when queries are received.

1) Graph construction: Graphs are created using GraphX [10] framework for graphs and graph-parallel computation embedded within the Spark distributed dataflow system. This framework benefits from in-memory processing capabilities of Spark, which allows exposing control over data partitioning and in-memory representation.

Further steps of the workflow are dedicated to retrieving the information such as social interaction types, hidden patterns in the graph and other insights.

2) Community detection: The information retrieving process starts from decomposing the graph into sets (communities) of densely inter-connected nodes in order to uncover hidden functional modules such as groups of users or topics in cross-media networks. The resulting meta-network, whose nodes are the communities, can be used to visualize the original network structure with reduced level of granularity.

Several algorithms have been proposed to detect communities in a network based on maximizing the benefit function known as modularity over possible divisions of a network [6], [7], [8]. In our system, we used the algorithm called Louvain that is known to outperform all other community detection methods in terms of computation time, while also maintaining a high quality level measured by the modularity [9].

The algorithm starts with assigning a different community to each node of the network. Based on the neighborhood search heuristic, the modularity is sequentially optimised for all nodes until no further improvement can be achieved. After this, a new network is built, whose nodes are the communities found during the neighborhood search process. These steps are iterated until there are no more changes and a maximum of modularity is achieved.

In order to efficiently deal with millions of nodes and tens of millions of edges, the DGAP applies a parallelized version of the Louvain algorithm taken from GraphX.

3) Authority and Hub calculation: To evaluate relationships between graph nodes, the Hyperlink-Induced Topic Search (HITS) algorithm was applied [10]. For each node of the graph, the algorithm calculates two metrics: *hub* and *authority*. The hub indicates how well a node is linked to other nodes, while the authority analyzes incoming links of a node. In other terms, the higher is the hub value, the better a node is pointed to other nodes (more outgoing links it has). The higher is the authority, the better a node is linked by other nodes (more incoming links it has).

In order to make the HITS algorithm applicable to large networks, we created a distributed version using Apache Spark DataFrame API and Scala. Our code is available in GitHub [11].

The rationale for using the HITS algorithm is that it gives both hub and authority scores for all the nodes in the graph, whereas, for example, PageRank is a centrality measure which mostly points to influential nodes based on random walk in the graph. In further research it is planned to integrate multiple ranking algorithms in order to get a comprehensive knowledge about the information contained in the links between nodes, as well as to allow switching between the algorithms depending on the size of the graph.

4) Role discovery: To then understand the role of each node within the network, we have implemented a distributed version of a state-of-the-art algorithm [11]. This algorithm is based on two metrics (*within-module degree* and *participation coefficient*) and provides for each node a role from a set of 7 roles: *ultra-peripheral, peripheral nodes, non-hub connectors,*

[10]https://spark.apache.org/graphx

[11]https://github.com/LianaN/hits-algorithm

non-hub kinless, provincial hubs, connector hubs, kinless hubs. The within-module degree (or z-score) z_i^d of node i calculates how well a node is connected within its community.

$$z_i^d = \frac{k_i^d - \bar{k}_{s_i}^d}{\sigma_{k_{s_i}^d}} \tag{1}$$

where d indicates the directionality of links (incoming or outgoing), k_i^d is the number of links between node i and other nodes in its community s_i, $k_{s_i}^d$ is the average of k over all the nodes in s_i, and $\sigma_{k_{s_i}^d}$ is the standard deviation of k^d in s_i.

The participation coefficient p_i^d of node i estimates how its links are distributed among all the different communities.

$$p_i^d = 1 - \sum_{s=1}^{N_M} \frac{k_{is}^d}{k_i^d} \tag{2}$$

where k_{is}^d is the number of links between node i and nodes in community s, k_i^d is the total degree of node i that corresponds to the directionality d, and N_M is the number of communities connected to node i, including its own community.

On the one hand, this set of discrete roles notably facilitates the understanding of elements of the complex network for non-technical users. On the other hand, as noted by [12], the algorithm in [11] was designed for non-directed networks while networks of interactions in cross-media networks are typically directed. Therefore, we split these metrics into two groups according the directionality d of edges: incoming and outgoing.

D. How to efficiently persist and query large graphs?

Once the metrics are estimated for all nodes of the graph with respect to different communication channels, the system proceeds to the storage of features of *communities*, *users* and *contents* in a distributed database.

For each community, we estimate the following features:

- CommunityId - the identifier of a community.
- CommunityName - the name of a community assigned according to the name of a node with the highest authority value (configurable from the front-end).
- NumberOfNodes - the number of nodes in the community.
- NumberOfEdges - the number of links between nodes of a given community.
- NumberOfPublications - the number of URL values in a community.
- PotentialAudience - the sum of audience values over nodes that belong to a community.
- AverageAuthority - the average of Authority values over nodes of a community.
- AverageHub - the average of Hub values over nodes of a community.
- EngagementRate - the number of posts in the community divided by the number of users in this community.
- StartDate - the date (UTC) when the first edge of a community was created.
- EndDate - the date (UTC) when the last edge of the community was created.
- StartDateTimezone - the timezone of the start date.
- EndDateTimezone - the timezone of the end date.

The users are characterized by thirteen features as follows:

- UserId - the identifier of a user.
- UserName - the name of a user.
- CommunityId - the identifier of a community.
- CommunityName - the name of a community assigned according to the name of a node with the highest authority value.
- NumberOfInfluencedUsers - the number of users influenced by a given user.
- NumberOfPosts - the number of posts published by a given user.
- RoleIn - the role of a user with respect to the incoming links.
- RoleOut - the role of a user with respect to the outgoing links.
- Audience - the audience of a user.
- Authority - the authority value of a user.
- Hub - the hub value of a user.
- InDegree - the number of incoming edges.
- OutDegree - the number of outgoing edges.
- EngagementRate - the number of posts published by a given user divided by the number of influenced users.

Finally the information of contents is structured as follows:

- URL - URL that defines a content.
- UserId - the identifier of the source node that has the earliest sharing date.
- UserLabel - the name of the source node that has the earliest sharing date.
- NumberOfShares - the number of edges that correspond to a given URL.
- NumberOfUsers - the number of users who shared a given URL.
- Audience - the sum of audience values of users who shared URL.
- FirstShare - the earliest date (UTC) when a given URL was shared.
- LastShare - the latest date (UTC) when a given URL was shared.
- FirstShareTimezone - the timezone of the earliest sharing date.
- LastShareTimezone - the timezone of the latest sharing date.

Once the data are stored in OrientDB, the HTTP POST request is sent to the end-point of API of the front-end in order to notify a marketer about the graph availability. Based on using the front-end application, the marketer can interact with the graph, query the information using filters, execute the animation to see how the selected content was diffused over the network or how the selected community was created, etc.

E. How to visualize large complex networks in an interpretable way?

Another important challenge solved by the DGAP platform is the visualization of large complex networks in an interpretable way. Once the graph mining algorithms are terminated and the results are stored in OrientDB, a cross-media graph becomes accessible through the front-end application. The front-end disposes of numerous filters that allow end-users querying the graphs stored in OrientDB. As a result, the front-end always shows only the sub-set of a large complex network according to the end-user filters. For example, instead of visualizing hundreds of thousands communities that may exist in the original graph, the end-user can filter communities by different parameters, such as authorities, hubs, audiences, datetime range, etc. Given the graph of communities, it is possible to navigate to the lower level that contains the graph of users of particular selected communities (see Figure 3).

To perform a more specific graph analysis, the DGAP platform allows end-users exporting graphs and sub-graphs into Graph Exchange XML format that is supported by Gephi and other graph analytics tools.

In the next section we explain the deployment of the platform.

V. Multi-machine cluster setup

The DGAP has been deployed on top of a cluster that was created using a free and open-source system for cloud computing called OpenStack. This system allows launching virtual machine instances by specifying virtual resource allocation profile (flavor), virtual hard disk file that is used as a template for creating a virtual machine (image), network, security group, key, and instance name.

The cluster consists of 7 instances, whose virtual resource allocation profiles are given in Figure 4. All instances are created using the CentOS 7 cloud image that includes conventional user name/password authentication and provides these credentials at the login prompt.

A floating IP address on the public provider network has been associated with each instance of the cluster. Moreover, we created the firewall rules to only allow access to the cluster through the TCP ports we opened.

Apache Ambari was used to automate the deployment and to facilitate the management of the Hadoop cluster that includes technologies such as Hadoop Distributed File System (HDFS) [12], YARN[13], Apache Hive[14], Apache Spark[15], and Apache Livy[16].

VI. Case study

In this section, we evaluate the performance of the DGAP platform on the cluster presented in section V, and compare the execution time of the workflow on different volume of raw data.

[12]https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[13]https://hortonworks.com/apache/yarn/

[14]https://hive.apache.org

[15]https://spark.apache.org

[16]https://livy.incubator.apache.org

Totally, three scenarios have been created using 10.000, 100.000 and 1.000.000 entries of the raw data set. In each scenario a new graph was generated using the corresponding raw data set stored in Elasticsearch. The numbers of nodes and edges in the graph were created based on a query presented in section IV.A. Instead of running a long-term Spark streaming job that listens to new queries, the platform launched the calculus on demand when a new input query was received by sending request to the Apache Livy REST web service. This approach considerably reduces the total cost of running big data cluster.

One of key challenges addressed by the DGAP platform was to efficiently persist and query the graphs representing cross-media networks that exceed the computation and memory capacities of a single processor. To solve this challenge, all the calculations were performed using Spark in a cluster mode on YARN, while the generated graphs were persisted in a distributed database OrientDB. The cluster mode supposes that the Spark driver runs inside an application master process managed by YARN on the cluster, while the client can go away after initiating the job.

The elapsed time of Spark operations was estimated using *spark.time* function. To properly estimate the time, it's important to materialize Spark DataFrame's by using *rdd.count*. However, this should not be included in the production code, as DataFrame's will be materialized twice. Finally, it should be noticed that the estimates of the elapsed time are approximate, because the physical execution query plan of Spark may change.

Table II: Elapsed time of major steps in the DGAP workflow

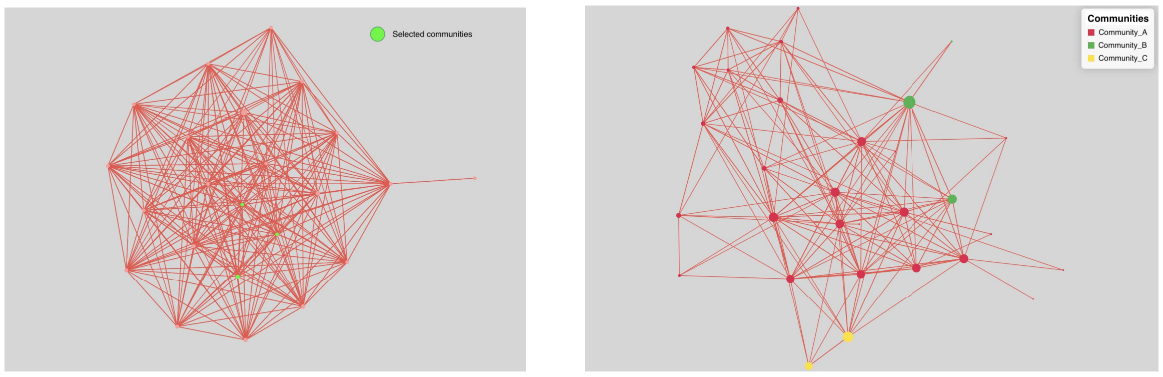
Operation	**Scenario**		
	10.000	***100.000***	***1.000.000***
Calculate communities	492.15	602.41	1400.67
Calculate authorities and hubs	26.45	41.75	191.33
Calculate roles	78.35	122.50	494.47
Calculate degrees	87.20	137.51	570.89
Save communities in OrientDB	650.25	746.73	1810.54
Save users in OrientDB	1278.47	1559.16	2779.20
Save contents in OrientDB	156.41	157.03	280.67

[a]Elapsed time is estimated in seconds

As shown in Table II, the elapsed time of operations increases with the growth of raw data set that is used for creating graphs. The most time-efficient operation refers to the HITS algorithm that calculates authorities and hubs among the nodes of the graph. For example, to calculate and assign the values of authority and hub metrics to graph nodes created based on the raw data set of 1.000.000 entries, 191.33 seconds were required. This algorithm was implemented in Spark by the authors of the paper and it can be found in the GitHub repository[17].

Furthermore, the calculation of roles seems to be very sensitive to the increase of a graph size. In particular, we may

[17]https://github.com/LianaN/hits-algorithm



(a) The sub-graph of communities (b) The graph of users of selected communities

Figure 3: The graph visualization after applying filters

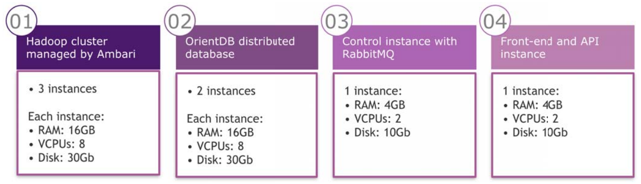


Figure 4: Virtual resource allocation in a cluster

observe the fourfold increment of elapsed time for the tenfold increase of raw data size from 100.000 to 1.000.000 entries.

The most time-consuming operation was the writing of data to the OrientDB database. To connect to the distributed database from Spark, we were using the $spark-orientdb$ connector repository[18]. Though other alternative approaches exist to write data to OrientDB in Spark, the given connector provided the best performance in terms of execution time. However, additional tuning of the writing process would be needed to reduce the corresponding elapsed time. Furthermore, allocating more resources to the Hadoop cluster and OrientDB distributed database would also boost this reduction.

A special attention has been dedicated to optimizing the Spark code of graph mining algorithms. It's a well known fact that Spark processes are lazy which means that the results are not estimated right away. The results are only computed when an action requires a result to be returned to the driver program. Therefore, we applied caching in cases when the lineage of the RDD branches out. For example, caching was used before running the iterative process of the HITS algorithm, in which case Spark keeps the source RDD around on the cluster for much faster access during each iteration.

VII. Conclusions

In this paper, we presented the innovative platform called DGAP that applies big data, graph mining and cloud computing in order to deliver a sophisticated quantitative and qualitative analytics toolkit for marketers. In contrast to many other existing media monitoring platforms, the DGAP supports distributed network analytics, thus helping marketers reveal social interaction types, hidden patterns in the network and the information diffusion over time.

Moreover, the distributed version of the HITS algorithm was developed and integrated into the platform. As demonstrated in the case study, this implementation of the algorithm showed a competitive performance on big amounts of data.

The DGAP platform uses Apache Livy REST web service in order to reduce the cost of running Spark applications in a cluster. Instead of having Spark streaming application running and consuming cluster resources, the platform allows executing big data analytics operations on demand.

The case study includes the performance investigation of the platform based on measuring elapsed time. This analysis, however, might be insufficient to generalize well to production use cases, because this approach often does not provide insights on why the system performs in a certain way. Therefore, further research includes the investigation of executor run time and CPU time metrics in order to analyze shuffling and disk reading/writing operations.

[18]https://dl.bintray.com/sbcd90/org.apache.spark

Acknowledgment

We would like to thank Jordi Roda (Eurecat) and Eduardo Rodríguez (Eurecat) for their valuable contribution to the development of this platform.

References

[1] T. L. Tuten and M. R. Solomon, *Social media marketing*. Sage, 2017.
[2] H. Sebei, M. A. Hadj Taieb, and M. Ben Aouicha, "Review of social media analytics process and big data pipeline," *Social Network Analysis and Mining*, vol. 8, no. 1, p. 30, Apr 2018. [Online]. Available: https://doi.org/10.1007/s13278-018-0507-0
[3] I. Stavrakantonakis, A.-E. Gagiu, H. Kasper, I. Toma, and A. Thalhammer, "An approach for evaluation of social media monitoring tools," *Common Value Management*, vol. 52, no. 1, pp. 52–64, 2012.
[4] B. Batrinca and P. C. Treleaven, "Social media analytics: a survey of techniques, tools and platforms," *AI & SOCIETY*, vol. 30, no. 1, pp. 89–116, Feb 2015. [Online]. Available: https://doi.org/10.1007/s00146-014-0549-4

[5] S. Beis, S. Papadopoulos, and Y. Kompatsiaris, “Benchmarking graph databases on the problem of community detection,” in *New Trends in Database and Information Systems II*, N. Bassiliades, M. Ivanovic, M. Kon-Popovska, Y. Manolopoulos, T. Palpanas, G. Trajcevski, and A. Vakali, Eds. Cham: Springer International Publishing, 2015, pp. 3–14.

[6] P. Pons and M. Latapy, “Computing communities in large networks using random walks,” *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218, 2006.

[7] A. Clauset, M. E. J. Newman, and C. Moore, “Finding community structure in very large networks,” *Phys. Rev. E*, vol. 70, p. 066111, Dec 2004. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.70.066111

[8] M. E. J. Newman, “Finding community structure in networks using the eigenvectors of matrices,” *Phys. Rev. E*, vol. 74, p. 036104, Sep 2006. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.74.036104

[9] V. Blondel, J. Guillaume, R. Lambiotte, and E. Mech, “Fast unfolding of communities in large networks,” *J. Stat. Mech*, p. P10008, 2008.

[10] J. M. Kleinberg, “Hubs, authorities, and communities,” *ACM Comput. Surv.*, vol. 31, p. 5, 1999.

[11] R. Guimera and L. A. N. Amaral, “Functional cartography of complex metabolic networks,” *Nature*, vol. 433, no. 7028, pp. 895–900, 2005. [Online]. Available: http://dx.doi.org/10.1038/nature03288

[12] S. González-Bailón, N. Wang, and J. Borge-Holthoefer, “The emergence of roles in large-scale networks of communication,” *EPJ Data Science*, vol. 3, no. 1, p. 32, Nov 2014. [Online]. Available: https://doi.org/10.1140/epjds/s13688-014-0032-y