Reference architecture for social networks graph analysis

Maxim Kolomeets[1,3,4], Amira Benachour[2,5], Didier El Baz[2,4],
Andrey Chechulin[1], Martin Strecker[3], and Igor Kotenko[1*]

[1]*Laboratory of Computer Security Problems, St. Petersburg Institute for Informatics and Automation (SPIIRAS), 199178, St. Petersburg, Russia*
{kolomeec, chechulin, kotenko}@comsec.spb.ru

[2]*LAAS CNRS, Université de Toulouse, 31031, Toulouse, France*
{abenachour,elbaz}@laas.fr

[3]*Université de Toulouse – IRIT, 31400, Toulouse, France*
martin.strecker@irit.fr

[4]*ITMO University, 197101, Saint-Petersburg, Russia*

[5]*University of Sciences and Technology Houari Boumediene, BP 32 El-Alia, 16111, Algiers, Algeria*

Received: October 3, 2019; Accepted: December 12, 2019; Published: December 31, 2019

Abstract

When analyzing social networks, graph data structures are often used. Such graphs may have a complex structure that makes their operational analysis difficult or even impossible. This paper discusses the key problems that researchers face in the field of processing big graphs in that particular area. The paper proposes a reference architecture for storage, analysis and visualization of social network graphs, as well as a big graph process "pipeline". Based on this pipeline it is possible to develop a tool that will be able to filter, aggregate and process in parallel big graphs of social networks, and at the same time take into account its structure. The paper includes the implementation of that pipeline using the OrientDB graph database for storage, parallel processing for graph measures calculation and visualization of big graphs using the D3 library. The paper also includes the conducted experiments based on the calculation of betweenness centrality of some graphs collected from the VKontakte social net.

Keywords: Social networks, graph analysis, big data, parallel computing, GPU, graph databases, data visualization, real-world networks.

1 Introduction

Social networks can give rise to various graph structures. They can be graphs of friends, subscribers, repost trees, trees of comments, dependency graphs of content and other. At the same time, big graphs often appear when analyzing even small amounts of source data. For example, if a small community is analyzed, it is necessary to take into account friends and maybe friends of friends. As we will show in subsection 4.1, if one will collect information about a community with 2000 participants and their friends, the graph can grow up to 770.000 vertices. And it can grow up much more if one needs information about friends of friends, relation to content or other information.

An investigation of big graphs implies many difficulties that are associated with the storage, processing and visualization of data – big graphs are at the core of the big data problematic. At the same

*Corresponding author: Laboratory of Computer Security Problems, St. Petersburg Institute for Informatics and Automation (SPIIRAS), 199178, St. Petersburg, Russia, Tel: +7-812-328-7181 time there are a lot of effective systems that are using big data for the analysis. Besides analysis systems for social networks there are many analysis software products for computer networks, mesh networks of Internet of Things, structures of algorithms, economic and political relationships, etc. The key moment is that big graph analysis is different from big data analysis – in graphs, an important component is the topology of the structure, which can make a significant contribution to the understanding of the data is analyzed. In current analysis systems of social networks are usually analyzing content and not considering analysis of structures.

A lot of other information sources also can be represented as a graph. The graphs that model these structures are often of great size and complexity. These occur in many domains like social simulation, e.g., road networks (US road network: 58,000,000 edges), social networks, e.g., big graph simulation (Twitter fellowship: 1,470,000,000 edges) and brain science, e.g., EU Brain project (Neural networks: 100,000,000,000 edges, 89 billion nodes). There are solutions for specific use-cases in the scientific literature and commercial software, but there is a lack of the general approach which will be able to combine various algorithms of graph preparation, processing and analysis.

This paper presents the concept of a tool for the analysis of big graphs of social networks that concentrates on the static structure analysis. The novelty of this work lies in generalization of the existing approaches for the various modules of the graph processing algorithms into one pipeline which includes all of the features. The scientific contribution of this work is the reference architecture for analysis of big graphs of social networks, which demonstrates the integration of various approaches into a single tool. Analysis methods include: graph analysis via graph algorithms, technologies for storing big graphs, graph visualization and graph processing based on parallel computing.

This article is a continuation of the previous paper [21] where we proposed an approach for creating the hierarchical graphs schemas of social networks.

The paper is organized as follows. In Section 2 we speak about the state of the art – the review of a systems that are using big graphs for the analysis and a review of areas where such systems are used. Section 3 is dedicated to the concept of analysis pipeline – a description of the proposed concept for big graph analysis and its parts. Section 4 describes the implementation – the "proof of concept" software prototype with the experiments based on the calculation of betweenness centrality measure for graphs of social networks. Section 5 is the discussion – evaluation of the concept and analysis of its utility. The last Section 6 concludes with the description of the results and plans for future work.

2 State of the art

Graphs can give valuable information about processes where the most important attribute is dependence of elements. Graph analysis objectives include but are not limited to analysis of streams and paths, structures that are formed by elements, elements' significance in relation to other elements and others. There are many areas where big graphs are necessary for the analysis.

For example transactions flows are used for detection of fraud in different systems from online commerce platforms [14] and banks [27] to cryptocurrency such as Bitcoin [16]. Using graphs it is possible to find the money flows and ultimate beneficiary of the transaction.

Dependency graphs are important for calculation of measures for specific users, that are later used in different recommendation services. For example, dependency graphs are used by such big companies as IMDB [9] and Netflix [18].

Journalists and police can use graphs for their investigation of fraud that is based on dependence between criminal and/or analyzed persons. For example International Consortium of Investigative Journalists (ICIJ) helped to reveal many tax evasion cases by building interactive graph visualization that showed dependence between offshore assets and dignitaries of different countries [17].

Also graphs can be used in smart transportation for evaluation and management of transport systems [26].

At the same time there are a lot of systems for social networks analysis. Social networks are classical targets for graph analysis that are used by companies that own social networks for advertising [23] and protection [22] of their reputation: for example, for detection of hate speech, prevention of artificial influence on society (for example, like cheating), detection of extremists, terrorists communities and bot farms. The Avalance Online [5] complex monitors media and social networks with the possibility of classifying the collected information, compiling a "dossier" for individuals using open data and determining the emotional coloring of messages. The features of this complex include the ability to build a graph of user connections, information for which is drawn from open sources. Another successful product that monitors user information on the Internet is Palantir [7]. This tool enables an analyst to analyze data from both open and closed sources, and the relationships between them. One of the specialized software for analyzing social networks is the "Oktopus" and "Zveroboi", which are used by experts in the Investigative Committee of the Russian Federation to investigate crimes [4]. VISR company provides service for child safety by analyzing with whom the child speaks in social networks [2]. For sociology social networks is also good environment for experiments and research. For example in [13] sociologists use machine learning for prediction of Destructive Stimuli based on the Ammon test and content analysis from social networks. Also, big companies that are own social networks such as Facebook, Instagram, Twitter, Reddit, VKontakte and others use information from their users for advertising.

Most social networks analysis tools are using content analysis with bound of machine learning and statistical methods. In that paper we concentrate on analysis of structures by graph algorithms.

Graph algorithms are distinguished in Network Science branch. That algorithm includes the calculation of centrality measures, finding the structures and clustering. At the same time it is complex to implement such algorithms on big graphs. Systems that analyse them using Network Science approaches need powerful subsystems that help to extract subgraphs, speed up calculations and visualize results. These subsystems form a pipeline, the concept of which we present in the next section.

3 Concept of analysis pipeline

The analysis of big graphs includes the tasks of both processing and storage of large volumes of data, and the tasks of analyzing the graph topology. For effective analysis of social networks, it is necessary to provide the following features:

1. Collection of data from social network.
2. Storage of a big graph that supports graph transformation.
3. Graph topology analysis using graph algorithms.
4. Parallel data processing that obtains results of analysis at an acceptable time.
5. Graph visualization that allows one to visually explore the results of the analysis.

Thus, to analyze big graphs one should use a set of technologies for collecting, gathering, storing, filtering, processing and visualizing data. On a high level they all can be united into a tool that looks like a "data pipeline". The concept of "pipeline" is presented in the Figure 1 and on the most abstract level it includes next main modules: data collection module, graph storage module, graph analysis module and visualization module. Also, each module includes smaller submodules.

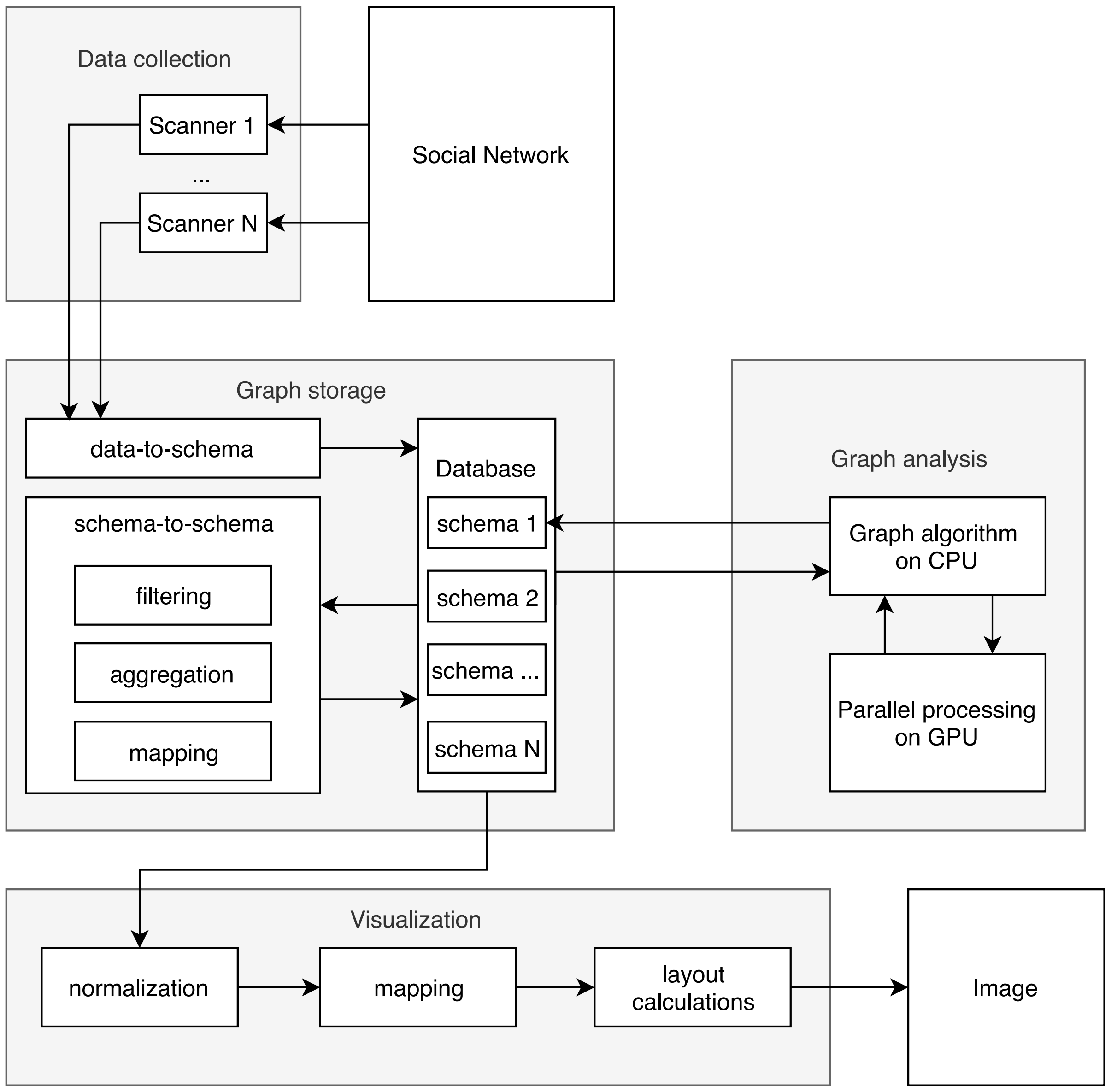


Figure 1: Proposed reference architecture in form of pipeline

3.1 Graph collection

Data collection can be carried out by a public or specialized method. A public method involves accessing HTML pages, parsing HTML markup, and extracting meaningful information. A specialized method use an API to access data.

Extracting data by parsing HTML pages can be an alternative to accessing the API functions in cases where the functions themselves are missing or when the social network imposes restrictions on the returned result (for example, “Twitter” limits the search for “tweets” for keywords to a 7-day interval). For automated loading of dynamic HTML pages and retrieving information, it is necessary to use libraries for HTML parsing.

Today, more and more social networks provide access to stored information through calls to Application Programming Interface (API) functions (including such big companies as Facebook, Instagram, Twitter, VKontakte, etc.). Access to source data using the API is legal, as the information providers already limit access to data that may legally be retrieved. The legality of the information collection can be also restricted according to the aim of data collection (usually it is allowed only for a personal usage and scientific purposes) and personal data protection laws (usually it can be solved through the usage of anonymization algorithms).

Scanners are used for data collection via making a request to the social network via an API. However, increasing the speed can trigger a block of scanner since social networks set a limit on the frequency of requests. For a high speed of data collection, a large number of simultaneously working scanners is required. Each scanner runs in its own virtual space - the thread of execution. So increasing the speed comes down to increasing the number of scanners that works in parallel.

3.2 Graph storage

A key component of graph storage is the ability to execute requests to retrieve parts of a graph. For example, the task of finding the friends of several people comes down to obtaining a subgraph based on topology – the possibility of finding the set of adjacent vertices (traversal function). The best storage implementation that takes into account topology is provided by graph databases. Also, the following properties of graph databases can be distinguished:

1. horizontal scaling – the ability to increase the number of operations per second due to the distribution of database resources on multiple devices;

2. multi-modal – the ability to store relationships between different types of data (including unstructured data that are common for social networks), such as documents;

3. schema – the possibility of implementing a case-dependent scheme (if one uses the modeling terminology - "meta model"), which is expressed in defining new types of vertices and edges of the graph. In addition, when storing big graphs, it becomes necessary to provide a flexible scheme - when adding new types of vertices and edges does not lead to changes in old records (it can happen in relational databases). The schema is necessary not only for retrieving new graphs, but also for retrieving graphs with structures that fit for analysis by algorithms and visualization (see Figure 1).

Such requirements help to filter and aggregate graphs faster and more easily. In general, aggregation and filtering is needed because it helps with solving the problem of graph transformation. Transformation can be reduced to:

1. bringing data to the structure of the database (data to schema);

2. combining several graphs into a single structure or creating representations that reflect only part of the original more complex graph (schema to schema).

"Data to schema" transformation means parsing of data to the database. During parsing one can aggregate and filter data for fitting data to a schema. It is pretty common for every database at the level of database filling and not unique.

"Schema to schema" transformations are more complex and unite different tasks of analysis:

1. graph filtering by:

    - the parameter of node or edge – for example, by "weight" of the vertex or edge (age of user, length of text and other);

- schema class of node or edge – for example, by specific class of the vertex or edge (user, post, comment and so on);
- by topology – for example, querying graph from some adjacent vertices of the vertices subset (retrieving friends, commentators, who liked and so on);

2. graph aggregation by

   - the parameter of node or edge – for example, by some categorical parameter of the vertex or edge;
   - schema class of vertex or edge – for example, by superclass of vertex or edge;
   - by topology – for example, by some path or specific sequences of traversal queries;

3. graph mapping to another schema – for example, by adding or removing the parameters, classes and superclasses.

Some of the graph transformations were considered in our previous paper [21], including aggregation by superclasses, aggregation by topological path that creates "virtual graphs" and filtering by class. Such transformation operations provide the ability to create new graphs.

As we said, graph transformations "data to schema" and "schema to schema" are different. The conversion of "data to schema" occurs at the stage of loading data into the database and should be provided using the appropriate API of DB that is based on the simple parsing of data.

A "schema to schema" conversion can occur in two ways. The first way is to use filtering, aggregating and mapping possibilities of the database itself using the database syntax or built-in functions as showed in [21]. The second way is to use graph algorithms, for example, graph algorithms can calculate some weights for the vertices after which can be performed filtering by weight. In this case, the task of transforming the "schema to schema" is partly transferred to the analysis module.

3.3 Graph analysis

The third element is a data analysis module that implements parallel graph processing. As we said, the key element of a graph is its topology. Thus, finding structures and determining the significance of the graph vertices can provide fundamentally new information about the analyzed data. There are many algorithms that can be distinguished: centrality, clustering, similarity, algorithms for finding the topological, potential-based algorithms in Laplace matrices and others.

Many centrality measures exist for quantifying the importance of a node in a network, utilizing both local and global information. Local measures as degree centrality and closeness centrality have linear time complexity, but are limited in their usefulness. Global measures such as betweenness centrality are of a much higher time complexity but are very useful.

We focused on the group of the algorithms that are based on the "Shortest Path Search", especially on the betweenness centrality measure.

Betweenness centrality is used to distinguish the most influential vertices in a graph. It is the ratio of shortest paths passing through a certain vertex v to the total number of shortest paths between all pairs of vertices. Let σ_{st} denote the number of shortest paths from a vertex $s \in V$ to a vertex $t \in V$. $\sigma_{st}(v)$ denotes the number of shortest paths from s to t that go through v. The betweenness centrality index of a vertex v is given by:

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1}$$

In social networks, it determines the individuals who influence the flow in the network. We consider a parallel betweenness centrality implementations based on Brandes' algorithm [12].

The key concept of Brandes's approach is the *dependency* δ of a vertex v with respect to a given source vertex s. We define $S(v)$ as the set of successor vertices of v.

$$\delta_s(v) = \sum_{w \in S(v)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_s(w)), \tag{2}$$

where σ_{sv} is the number of shortest paths between nodes s and v and $\delta_s(w)$ is the *dependency* of vertex w with respect to s.

The recursive relationship between the *dependency* of a vertex and the *dependency* of its successors allows a more efficient calculation of the betweenness centrality. Brandes's algorithm splits the betweenness centrality calculation into two major steps:

1. Find the number of shortest paths between each pair of vertices
2. Sum the dependencies for each vertex

Thus, *BC* measures are redefined as follow:

$$BC(v) = \sum_{s \neq v} \delta_s(v) \tag{3}$$

3.4 Graph Visualization

The third element is a visualization module that normalizes and maps the results into graphic primitives (color, size, transparency, etc.) and renders the image.

The final result is visualization. For example, the graph with weights that have been calculated by the graph analysis module, can be presented as a 3D graph where the vertex weight will be expressed as the color of the vertex. The basic submodules of visualization are:

1. normalization – includes the selection of the right scale;
2. mapping – is bringing the values to visual, for example, bringing number to color or volume;
3. layout calculation – is calculation of the positions of nodes depending on the layout (tree, circular, force-layout, etc.).

Normalization is necessary because data produced by centrality algorithm can be distributed differently. For example, a large number of nodes can have small centrality measure while a small number of nodes can have large centrality value. Without normalization, the difference between extremal values is so small that an analyst will not see any difference. For normalization are usually used scales that depends on the the distribution of values: linear scale, pow scale, log scale and others.

Mapping is the process when numerical or categorial values transforms to visual values. The visual values can be: size, shape, color, hue, saturation, opacity, texture, scaling speed, movement speed and rotation speed. For example vertex can be a cube and its rotation speed can express the centrality. For scaling speed, movement speed and rotation speed and opacity mapping is normalization from 0 to 1, because speed and opacity can be expressed as percentage from minimum for maximum. Shape and mapping can be defined only for categorial values, so mapping is the setting of specific shape or texture for each category. For color, hue and saturation mapping is the normalization of numerical value to RGB (red, green and blue value), where minimum value is setting as one color and maximum value as other color. For size it is important to take into account the shape. For example if cube is defined through length of side, so size should be calculated using the formula of dependence of side length on volume. It is important to note that the rotation speed also depends on the shape, and it can not be applied to sphere.

Layouts calculation is the calculation of the position of nodes in order to make a graph more easily to perceive. There are many layout methods [19] but the biggest part of them is based on the force model. For force layout it is necessary to set the charge for vertex (analog of electron – define the force with which the vertices of the graph will be repelled from each other) and length of edge (analog of spring – defines the spring's ability to stretch).

Having the visual values and coordinates of the vertices one can render a graph to the image that will be the result of processing.

4 Implementation

We implemented the proposed concept and made experiments with the analysis of the betweenness centrality of graphs. In the implementation we considered 5 datasets of different sizes and edge density. The goal is to find the leaders of the community and evaluate how many friends they have. It is also interesting to find the sub-communities and evaluate their size. Th implementation of the pipeline includes 4 main modules – collection of data, graph storage, analysis with GPU and visualization.

4.1 Data collection

As data source we select the social network VKontakte, because it has an open API [8] for collection of public data.

In VK users can make their profiles private, so it is impossible to get any information about such accounts.

We collect data about users of one group "Go SlackLine Team" – a community of fans of walking on a sling [3]. The community includes only 2104 users, but when we scan their friends we got 779020 users (around 370 friends per user in average). The collection took 112 minutes with 3 scanners that worked in parallel. This dataset contains many private profiles that will be filtered later.

Another, random data set, contains 402.000 users with open profiles who have at least 2 friends or subscribers. The collection took 215 minutes with 3 scanners that worked in parallel. It is important to note, that the second dataset was loaded more slowly than the first one because in the second case we loaded only open profiles with more data, while the first dataset includes many private profiles.

4.2 Graph storage

As a storage we select the graph database OrientDB [6], as it provides all required functionality and using SQL like syntax for querying. Also, OrientDB includes built in traversal function, web server that helps to implement necessary API and can be scaled to multiple servers for performance increase. As a database schema we select a schema that was described in our previous paper: it is the hierarchical schema for social networks that can aggregate or filter using classes and superclasses of the scheme hierarchy [21].

So, the data pipeline in graph storage module is organized as follows.

The "Data to schema" submodule is implemented on Node.js with the help of the OriendDB API – it is getting data from VKontakte API and brings graph to the schema of database. Using the built in functions, database syntax and Node.js we implement the "schema to schema" submodule that supports the and filtering of edges and vertices of the graph by property, class and superclass. It is also possible to traverse set of vertices. Using “schema to schema” submodule we are bringing data to the schema that is necessary for GPU analysis and for visualization.

For both the “Go SlackLine Team” and “Random data” graphs we aggregate “friend” and “subscribers” edges by one to get the relations between users.

Also for the first graph we made filtering by property “number of friend”: we create 4 new graphs, where where deleted users who have less than 5, 4, 3 and 2 users. This was done because the first graph contains many users with only one “friend” edge – it happened because of the many private profiles. Vertices with one edge form a star structures that is not useful for the analysis, so filtering removes them. That’s why we also collect another graph that has a higher edge density and less star structures to test performance on highly connected graph.

4.3 Graph analysis

4.3.1 Parallel processing using GPU

Meaningful information concerning graphs may be difficult to extract without applying specific graph algorithms. However, it is important to obtain results within acceptable times. For that purpose it can be interesting to consider parallel processing on GPU devices.

A GPU consists of many individual Streaming Multiprocessors (SMs), each of which executes in parallel with the others. Each SM contains a set of cores, called Stream Processors (SPs) that share control logics and instruction caches.

The GPU includes a large amount of device memory with high bandwidth. Each SM has a low latency *shared memory* and is also attached with a read-only texture and constant caches. On each SM, there are also compiler-managed registers used as a local storage for each thread.

To best support graphics processors NVIDIA Corporation introduced Compute Unified Device Architecture (CUDA) a platform for a parallel computing.

In the CUDA programming model, a program consists of multiple phases that are executed on either the CPU or the GPU. Parallel functions executed on the GPU are called *Kernels*. When launched, a CUDA kernel is executed N times in parallel by N different CUDA threads. These threads are organized into a three-level hierarchy. All threads generated by the kernel are collectively called a grid, which is an array of thread blocks.

Due to such massive amounts of parallelism, GPUs are attractive platforms to accelerate computations for many applications (see [11],[24],[25]) and in particular graph processing [15].

4.3.2 Graph algorithms

As explained in section 3.3 we are interested in accelerating centrality algorithms like betweeness centrality using GPUs. Brandes’s implementation performs two phases to calculates BC scores:

1. A parallel forward BFS traversal to accumulate sigma (σ) values for each node;

2. A parallel backward BFS to compute centrality values based on the results of the first phase.

The parallel algorithm was implemented on a single node GPU NVIDIA Tesla K40 that has 2880 CUDA Cores and 12GB of GDDR5 Memory and is a CUDA compute capability 3.5 GPU (Kepler architecture) with CUDA 8.0 toolkit. The CPU is a 4 cores Intel Xeon E5640 processor running at 2.67 GHz with a 12 MB cache and 12 GB of DRAM. Initial results on a set of graphs are shown in Table 1.

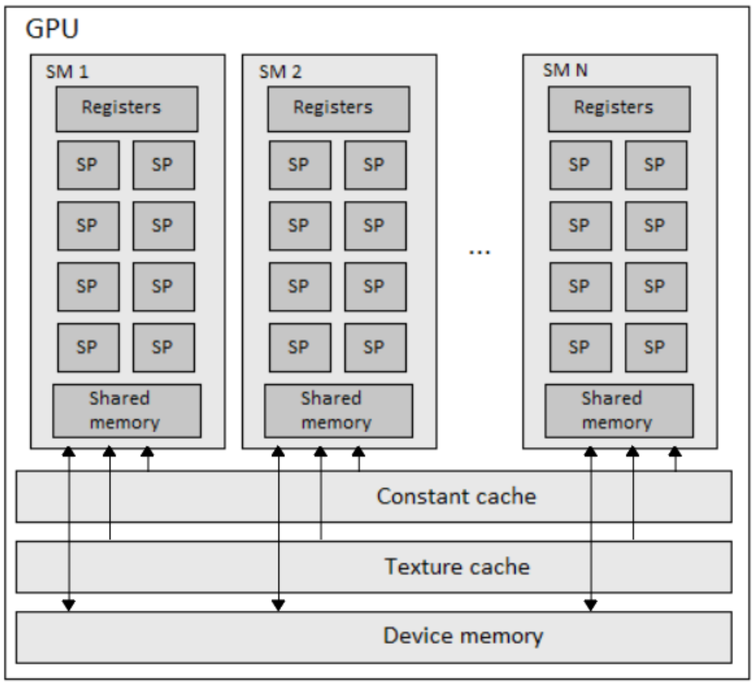


Figure 2: GPU memory architecture

Algorithm 1: GPU betweenness centrality

```
   Input: Graph G
   Output: $BC[]$
1  Initialisation:
2  foreach $v \in V$ in parallel do
3      if $v = s$ then
4          $\sigma(v) \leftarrow 0$
5      else
6          $\sigma(v) \leftarrow \infty$
7      $\delta(v) \leftarrow 0$
8  Forward BFS: accumulate sigma values
9  foreach $v \in V$ in parallel do
10     Compute $\sigma(v)$ and Pred$(v)$
11 Backward BFS: accumulate dependency values
12 foreach $v \in V$ in parallel do
13     Compute $\delta(v)$
14     $BC[v] += \delta(v)$
15 return $BC[]$
```

Dataset	Nodes	Edges	Time CPU (msec)	Time GPU (msec)
remove_2	9842	61438	80373	28296
remove_3	4919	32020	14967	13983
remove_4	3038	17060	3886	7764
remove_5	2187	8706	1021	5325
random	402916	9850186	+24H	5765261

Table 1: Execution times

4.4 Graph visualization

For graph visualization we used D3.js library [10]. Betweenness centrality measure is expressed by color and size of the vertex – double expression helps to percept value more accurately.

For size we use power scale non-normalized betweenness centrality. As the size of the vertices in D3 is defined by radius of the sphere, we map volume V (that is equal to centrality) into radius r by next formula

$$r = \sqrt[3]{\frac{3V}{4\pi}}. \tag{4}$$

For color we use linear scale no normalize betweenness centrality. The color value was mapped into RGB value from blue to red (Figure 3).

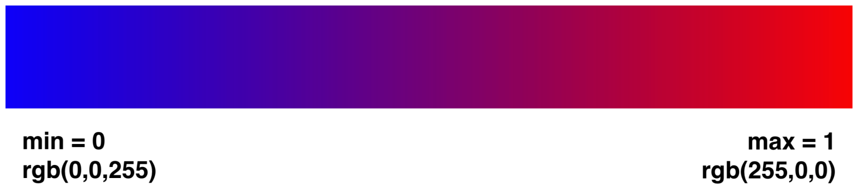
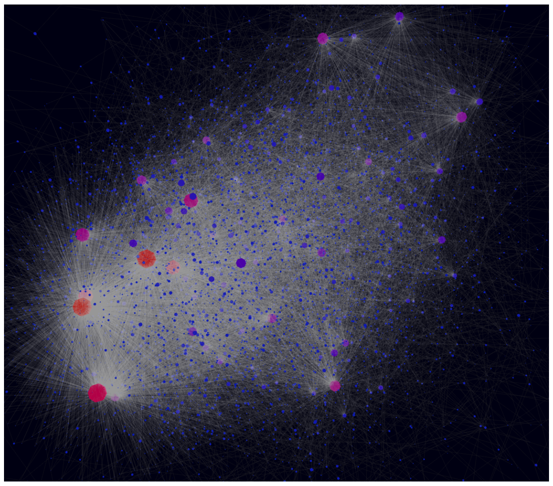
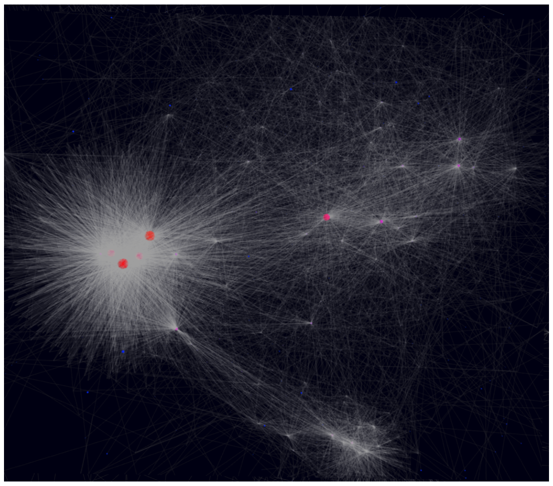


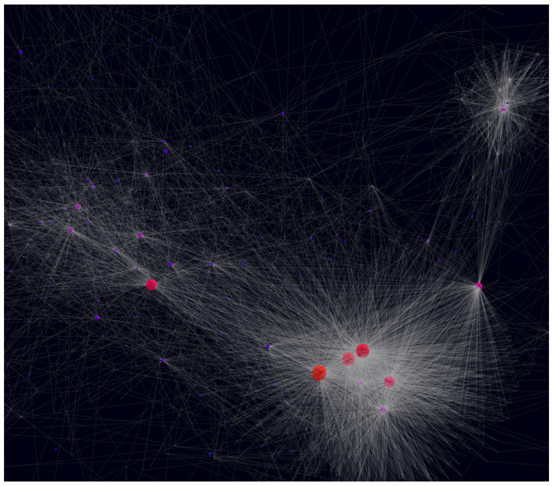
Figure 3: Color mapping that was used for betweenness centrality

As the layout we took 3D force layout algorithm that is based on the charge of vertices (the force with which the vertices of the graph repel) and strength of edges (edge stretching ratio). For force layout we calculate values of charge and strength individually for every vertex and edge based on the betweenness centrality and degree of each vertex (number of ingoing and outgoing edges for each vertex) as presented in [20]. We set values according to the following rules:

1. For vertices that have only one edge and for that edge:
   (a) to set small charge value for vertex;
   (b) to set maximum strength value of 1 for edge (the edge cannot be stretched);
   (c) to set a small edge length.
2. For vertices that have the biggest betweenness centrality (top 5%) and for edges between these vertices:
   (a) to set a big charge value for vertex;
   (b) to set maximum strength value of 1 (the edge cannot be stretched) for edges between these vertices (having the largest number of edges).


(a) remove_2 dataset – users who have at least 2 friends


(b) remove_3 dataset – users who have at least 3 friends


(c) remove_4 dataset – users who have at least 4 friends

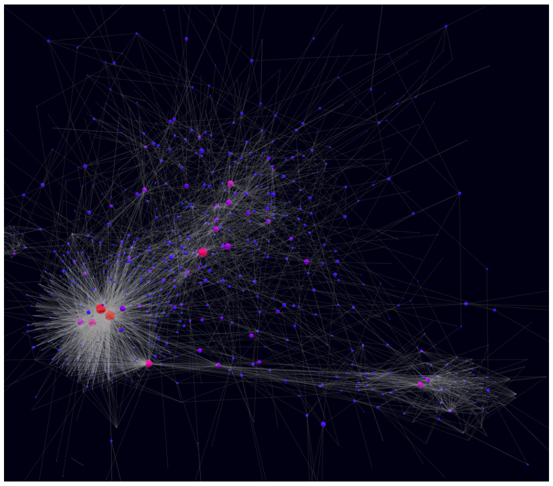

(d) remove_5 dataset – users who have at least 5 friends

Figure 4: Visualisation of "Go SlackLine Team" community with betweenness centrality expressed by size of the vertex, color of vertex and and force-layout parameters. Big red nodes have big values of betweenness centrality, while small blue nodes have small values.

   (c) to set a big edge length.

3. For all other vertices and edges:
   (a) to set an average charge value for vertex;
   (b) to set value of 0.1 for edge (the edge can be strongly stretched);
   (c) to set an average edge length.

As a result, vertices with highest betweenness centrality will be far from each other and will form a "skeleton" of the graph. The vertices with only one edge will be grouped into clouds. Other vertices will be between the most central vertices.

The visualization of datasets from table 1 is presented in Figure 4 and Figure 5 and via the link [1]. For dataset with random users we filter graph to 7000 nodes with biggest centrality.

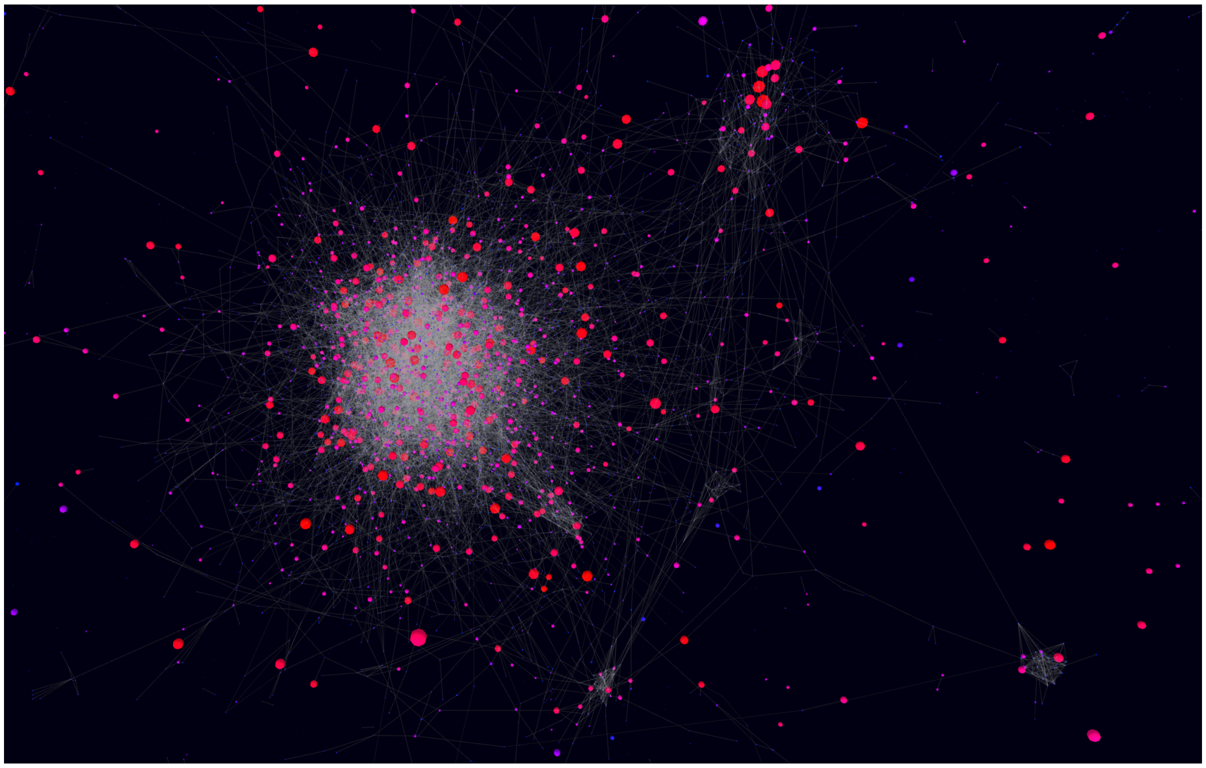


Figure 5: Dataset of 402916 users that was reduced to 7000 users with biggest centrality measure

The big red nodes indicate the leaders of the community. Highlighting by color and size helps to manually select these users if needed. It is also possible to identify sub-communities that are formed around the most central nodes.

5 Discussion

As we see, analysis of structure in social networks require different solutions for every stage of analysis. They can be united in a single pipeline that can make such analysis automatic.

The biggest challenge is increasing the speed of each part of data pipeline to get results in acceptable time. The decisions based on scalable graph databases and usage of GPUs for data processing can significantly reduce time of analysis.

Modern graph storages can perform basic manipulations with graphs by their own, including graph transformations to the specific schemas. Such an approach helps to reduce the needs of data transferring between modules of storage and analysis for simple operations, such as filtering and aggregation.

From Table 1, we note that computing accelerators GPUs permit one to reduce computing time when the graphs are sufficiently large, e.g., remove_2 dataset and random dataset. In the particular case of remove_2 dataset we observe that we obtain a speed up equal to 2.84x. In the case of random dataset the speed up is more important and we have stopped the computations on the CPU after 24H. This is due to the fact that thread occupancy on GPU is more important for these two graphs.

Graph visualization is the result of the analysis. Perception is increased by metrics expression. For example, on small graph 4d it is possible to locate central nodes by graph topology even without coloring by centrality. But when graph is growing up to the structure as in Figure 4a, it is becomes complicated to do that without the help of node coloring.

6 Conclusion

In the paper we present the reference architecture based on the data pipeline for the analysis of the graphs of social networks. Based on this pipeline it is possible to develop a tool that is able to filter, aggregate and process big graphs of social networks, and at the same time takes into account its structure. We present data collection, graph storage, graph analysis by GPU and graph visualization modules for that pipeline. We implement that pipeline and conducted experiments based on the calculation of betweenness centrality of the graphs in real-world social network.

Possible applications of the proposed architecture are not only the analysis of social networks; it can also be used for analyzing big and unstructured graphs in other areas. In future work we plan to extend the current implementation of the pipeline architecture so as to take into account other centrality measures. We also will investigate the possibilities of speed up the calculations for centrality measures, make research in visualization of big graphs based on the graph embeddings and will provide a mechanism for visually managing the analysis process.

Acknowledgment

Parts of this study (Sections 3.3 and 4.3) has been made possible by a funding of PHC Tassili Bigreen 40160QB – Algorithms of large data graphs on distributed memory architecture. Parts related to the social networks (Sections 2, 3.1, 3.4, 4.1 and 4.4) was supported by the grant of RSF #18-71-10094 in SPIIRAS. Parts related to the big data storage (Sections 3.2 and 4.2) are carried out by the budget (the project No. 0073-2019-0002).

References

[1] "Graphs visualization," http://comsec.spb.ru/files/jowua_ref_arc/index.html [Online; accessed on December 15, 2019].

[2] "Visr company," https://visr.co/#home [Online; accessed on December 15, 2019].

[3] "Go slackline team community in vk," https://vk.com/vgost [Online; accessed on December 15, 2019], 2006.

[4] "Baltinfocom company," http://baltinfocom.ru/BigData [Online; accessed on December 15, 2019], 2007.

[5] "Avalance online software," https://avl.team [Online; accessed on December 15, 2019], 2019.

[6] "Orientdb," https://orientdb.com/ [Online; accessed on December 15, 2019], 2019.

[7] "Palantir company," https://www.palantir.com/solutions/ [Online; accessed on December 15, 2019], 2019.

[8] "Vkontakte," https://vk.com/dev/methods [Online; accessed on December 15, 2019], 2019.

[9] R. Angelova and G. Weikum, "Graph-based text classification: learn from your neighbors," in *Proc. of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR'06), Seattle, Washington, USA*. ACM, August 2006, pp. 485–492.

[10] M. Bostock, "D3 data-driven documents," https://d3js.org/ [Online; accessed on December 15, 2019], 2019.

[11] V. Boyer, D. El Baz, and M. Elkihel, "Solving knapsack problems on GPU," *Computers & Operations Research*, vol. 39, no. 1, pp. 42–47, January 2012.

[12] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, August 2001.

[13] A. Branitskiy, D. Levshun, N. Krasilnikova, E. Doynikova, I. Kotenko, A. Tishkov, N. Vanchakova, and A. Chechulin, "Determination of young generation's sensitivity to the destructive stimuli based on the infor-

mation in social networks," *Journal of Internet Services and Information Security (JISIS)*, vol. 9, no. 3, pp. 1–20, 2019.

[14] J. Chen, Y. Tao, H. Wang, and T. Chen, "Big data based fraud risk management at Alibaba," *The Journal of Finance and Data Science*, vol. 1, no. 1, pp. 1–10, December 2015.

[15] A. Dabah, A. Bendjoudi, A. AitZai, D. El-Baz, and N. N. Taboudjemat, "Hybrid multi-core CPU and GPU-based b&b approaches for the blocking job shop scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 73–86, July 2018.

[16] G. Di Battista, V. Di Donato, M. Patrignani, M. Pizzonia, V. Roselli, and R. Tamassia, "Bitconeview: visualization of flows in the bitcoin transaction graph," in *Proc. of the 2015 IEEE Symposium on Visualization for Cyber Security (VizSec'15), Chicago, Illinois, USA*. IEEE, October 2015, pp. 1–8.

[17] D. Dominguez, O. Pantoja, and M. González, "Mapping the global offshoring network through the panama papers," in *Proc. of the 2018 International Conference on Information Theoretic Security (ICITS'18), Libertad city, Ecuador*. Springer, Cham, January 2018, pp. 407–416.

[18] E. Elahi and Y. Raimond, "Mlconf seattle 2015, Spark and GraphX in the Netflix recommender system," https://www.slideshare.net/ehtshamelahi/ml-conf-seattle-2015 [Online; accessed on December 15, 2019], May 2015.

[19] Y. Hu, "Algorithms for visualizing large networks," *Combinatorial Scientific Computing*, vol. 5, no. 3, pp. 180–186, September 2011.

[20] M. Kalameyets, A. Chechulin, and I. Kotenko, "The technique of structuring social network graphs for visual analysis of user groups to counter inappropriate, dubious and harmful information," in *Proc. of the II International Scientific and Practical Conference "Fuzzy Technologies in the Industry" (FTI'18), Ulyanovsk, Russia*. Ulyanovsk State Technical University, October 2018, pp. 116–125.

[21] M. Kolomeets, A. Chechulin, and I. Kotenko, "Social networks analysis by graph algorithms on the example of the VKontakte social network," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 10, no. 2, pp. 55–75, June 2019.

[22] I. Kotenko, I. Saenko, A. Chechulin, V. Desnitsky, L. Vitkova, and A. Pronoza, "Monitoring and counteraction to malicious influences in the information space of social networks," in *Proc. of the 10th International Conference on Social Informatics (SocInfo'18), St. Petersburg, Russia*, ser. Lecture Notes in Computer Science, vol. 11186. Springer, Cham, September 2018, pp. 159–167.

[23] Y. Li, B. Q. Zhao, and J. Lui, "On modeling product advertisement in large-scale online social networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1412–1425, December 2012.

[24] J. Luo, S. Fujimura, D. El Baz, and B. Plazolles, "GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 244–257, November 2019.

[25] B. Plazolles, D. El Baz, M. Spel, V. Rivola, and P. Gegout, "Simd monte-carlo numerical simulations accelerated on GPU and xeon phi," *International Journal of Parallel Programming*, vol. 46, no. 3, pp. 584–606, May 2017.

[26] M. Rathore, A. Ahmad, A. Paul, and U. K. Thikshaja, "Exploiting real-time big data to empower smart transportation using big graphs," in *Proc. of the 2016 IEEE Region 10 Symposium (TENSYMP'16), Bali, Indonesia*. IEEE, May 2016, pp. 135–139.

[27] G. Sadowski and P. Rathle, "Fraud detection: Discovering connections with graph databases," neo4j, Tech. Rep., 2014.

Author Biography

Maxim Kolomeets is currently a PhD student under joint supervision of ITMO University and University Toulouse 3 Paul Sabatier. He junior researcher at the Laboratory of Computer Security Problems of St.Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). His research interests include distributed system security, and security visualization. He is the author of more than 20 refereed publications.

Amira Benachour received the Master degree in software engineering from University of Blida 1, Algeria in 2017. She is presently pursuing her Ph.D. degree at the Laboratory of Analysis and Systems Architecture (LAAS-CNRS) Toulouse France and University of Sciences and Technology Houari Boumedienne, Algiers. Her Ph.D. is on graph processing and parallel computing.

Didier El Baz received the Engineer degree in Electrical Engineering and Computer Science from National Institute of Applied Sciences (INSA) in Toulouse France in 1981 and the Doctor Engineer degree in Control Theory from INSA Toulouse in January 1984. Dr. El Baz was visiting scientist in the Laboratory for Information and Decision Systems, MIT Cambridge Massachusetts, USA, in 1984. He received the HDR from INP Toulouse in 1998. He is the founder and head of the Distributed Computing and Asynchronism team at the Laboratory of Analysis and Systems Architecture of CNRS (LAAS-CNRS), Toulouse. Dr. El Baz is the author of 40 papers in referred international journals and 70 papers in referred international conferences. His fields of interest are in HPC, parallel and distributed computing, applied mathematics and cyber-physical systems. Dr. Eng. Didier El Baz is presently co-head of the international laboratory Information Security of Cyber-Physical Systems, St. Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), Russian Federation.

Andrey Chechulin received his B.S. and M.S. in Computer science and computer facilities from Saint-Petersburg State Polytechnical University and PhD from St.Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS) in 2013. In 2015 he was awarded the medal of the Russian Academy of Science in area of computer science, computer engineering and automation. At the moment he holds a position of leading researcher at the Laboratory of Computer Security Problems of SPIIRAS. He is the author of more than 70 refereed publications and has a high experience in the research on computer network security. His primary research interests include computer network security, intrusion detection, analysis of the network traffic and vulnerabilities.

Martin Strecker obtained a diploma in computer science from the Technical University of Darmstadt and the INP Grenoble. He then worked as teaching and research associate at the University of Ulm, where he earned a PhD degree on the topic of program and proof development in Type Theory. After a post-doc at the TU München, he joined the University of Toulouse as Associate Professor. Martin's main interests are in verified program development, semantics of programming languages, proof assistants and logics in computer science.

Igor Kotenko graduated with honors from St.Petersburg Academy of Space Engineering and St. Petersburg Signal Academy. He obtained the Ph.D. degree in 1990 and the National degree of Doctor of Engineering Science in 1999. He is Professor of computer science and Head of the Laboratory of Computer Security Problems of St. Petersburg Institute for Informatics and Automation. He is the author of more than 500 refereed publications, including 13 textbooks and monographs. Igor Kotenko has a high experience in the research on computer network security and participated in several projects on developing new security technologies. For example, he was a project leader in the research projects from the US Air Force research department, via its EOARD (European Office of Aerospace Research and Development) branch, EU FP7 and FP6 Projects, HP, Intel, F-Secure, etc. The research results of Igor Kotenko were tested and implemented in more than fifty Russian research and development projects.