Visualizing Large Knowledge Graphs: A Performance Analysis

Juan Gómez-Romero [a,b], Miguel Molina-Solana [b], Axel Oehmichen [b], Yike Guo [b]

[a] Department of Computer Science and Artificial Intelligence, University of Granada, Spain
[b] Data Science Institute, Imperial College London, United Kingdom

E-mail
jgomez@decsai.ugr.es
m.molina-solana@imperial.ac.uk
axelfrancois.oehmichen11@imperial.ac.uk
y.guo@imperial.ac.uk

Corresponding author
Juan Gómez-Romero
Department of Computer Science and Artificial Intelligence
University of Granada
jgomez@decsai.ugr.es
C/ Periodista Daniel Saucedo Aranda s/n
18071 Granada (Spain)

Manuscript submitted to Future Generation Computer Systems:
https://www.sciencedirect.com/science/article/pii/S0167739X17323610

Visualizing Knowledge Graphs: A Performance Analysis

Abstract

Knowledge graphs are an increasingly important source of data and context information in Data Science. A first step in data analysis is data exploration, in which visualization plays a key role. Currently, Semantic Web technologies are prevalent for modelling and querying knowledge graphs; however, most visualization approaches in this area tend to be overly simplified and targeted to small-sized representations. In this work, we describe and evaluate the performance of a Big Data architecture applied to large-scale knowledge graph visualization. To do so, we have implemented a graph processing pipeline in the Apache Spark framework and carried out several experiments with real-world and synthetic graphs. We show that distributed implementations of the graph building, metric calculation and layout stages can efficiently manage very large graphs, even without applying partitioning or incremental processing strategies.

Keywords
graphs; visualization; big data; linked data; performance analysis

1 Introduction

Visualization is a fundamental task in the scientific discovery process, although it was not until the late 80s when it was properly established as a research area [1]. Leigh et al. stressed that visualization serves at least three important purposes [2]: (i) quick verification of simulation models; (ii) quick delivery of simulation results; and (iii) easier communication to a layman. In Data Science, visualization is particularly valuable in the exploration and presentation of the results stages; that is, at the beginning and at the end of the data analysis process [3]. These are two phases which require a transfer of knowledge between data scientists and domain experts. Indeed, data scientists need to validate their assumptions and findings with the experts in order to redesign, repeat, or conclude the process.

Visual data exploration is particularly useful when little is known about the source data and the analysis goals are vague. In this context, visual data exploration can be viewed as an evolving hypothesis-generation process [4], during which hypotheses can be validated or rejected on a visual basis, and new ones can be introduced. According to Shneiderman [5], visual data exploration is a three-step procedure encompassing data overview, facet zoom and filter, and on-demand detail request.

Knowledge graphs —and more specifically, Semantic Web ontologies and linked data— are an increasingly important source of primary and contextual data in Data Science [6]. The rationale is that Semantic Web technologies provide a suitable infrastructure for publishing, storing, retrieving, reusing, integrating, and analyzing data [7]. Linked data languages are based on 3-tuple representations; specifically, RDF (Resource Description Framework) uses *<subject, predicate, object>* tuples [8], and OWL (Ontology Web Language) add formal semantics to RDF triples, allowing the definition of more complex hierarchies and networks [9]. Therefore, they have favored the development of graph-based visualizations. Other visualizations, such as topic maps [10], geographic maps [11], and statistical charts [12] have also been used when the semantics of the underlying data is appropriate.

The growth of available data is a common issue in Data Science, Semantic Web and Data Visualization, and so the necessity of frameworks capable of dealing with this overabundance. Regarding visualization, hardware and software tools are being created to deal with large-scale datasets and to visualize them on large screens and video-walls. Visual spaces, supported by

distributed data processing clusters, are fostering speed, accuracy, comprehension and confidence on the data analysis. Early works in this topic were developed by the Electronic Visualization Laboratory at the University of Illinois at Chicago, resulting in the CAVE [13] and CAVE2 [14] environments. Other examples of visualization environments have been described in the literature [2,15,16]. Such proposals have been shown effective in domains in which large amounts of intensively-interrelated data are generated. However, the increasing potential of new Big Data technologies for distributed data collection and processing calls for new initiatives.

Recently, our research group proved that graph visualizations, backed by a large-scale visualization environment, can be very helpful to understand the dynamics of different phenomena; e.g. Bitcoin transactions [17] and online discussions [18]. In these works, we showed how filtering, grouping and highlighting specific patterns help to increase the usability of large graph visualizations. Nevertheless, we also experienced that the scalability of the implemented platforms is limited. Furthermore, until recently we lacked a clear characterization of the efficiency of the graph generation process, and a wider analysis of the Big Data tools that can help to mitigate performance issues.

This paper presents the results of evaluating the computational performance of large-scale graph processing technologies for different visualization tasks. Such research work is an open challenge in this area [19,20]. Our analysis focuses on the performance of the data preparation, calculation of graph metrics, and graph layout stages for different graph types. For the experiments, we developed a software package implementing these tasks based on GraphX [21], the API included in Apache Spark [22] for graphs and graph-parallel computation.

The main contributions of the paper are the following:

- We describe a Big Data architecture for the generation of customized knowledge graph visualizations.
- We perform several experiments with different freely available datasets, in order to assess the efficiency of the graph visualization process.
- We conclude that the performance of the graph building and the graph layout stages significantly improves in the Big Data setup, allowing us to process graphs with more than one billion edges.

In this work we do not consider the performance of graph retrieval from permanent storage — evaluations of graph databases and linked data engines can be found in the literature [23,24]. We also leave out of the scope of the paper the study of the usability of the resulting graphs, for which we refer the reader to [25].

The remainder of the paper is structured as follows. In the next section, we review some related works on visualization of linked data and ontologies by means of graphs. In Section 3, we propose a generic graph visualization pipeline, an architecture based on Big Data technologies, and an implementation. In Section 4, we perform several experiments with graph extracted from different data resources: DBPedia knowledge graphs, synthetic graphs, and graphs from the Stanford Large Network Dataset Collection. Next, in Section 5 we discuss the results, highlight the main outcomes and identify open problems. The paper finishes with a summary of the conclusions and a description of prospective directions for future work.

2 Related work

Linked data visualization techniques date as early as ontology languages. Predecessors of computable ontologies –such as semantic networks, dependency graphs, and frames– provide a

graphical notation, which can be seen as an initial attempt to visualize knowledge graphs. In this regard, Sowa's conceptual graphs bring together the interpretability of visual semantic networks and the formal underpinnings of first-order logic [26]. Description Logic languages, considered the de facto standard ontology languages, also pursue the goal of providing well-defined semantics for expressive ontologies [27]. For the purpose of this paper, we will assume Linked Data and Semantic Web ontologies as the most recent incarnation of knowledge graphs.

Simpler models in the RDF (Resource Description Framework) and RDF-Schema languages, not bound to a Description Logic, do not entail complex semantics, and therefore they can be directly depicted as directed graphs [8]. Linked data has been typically visualized in this way. In contrast to RDF, the depiction of OWL (Ontology Web Language) models is not straightforward because of its capability to define complex relationships [9]. The OWL language specification does not state a graphical representation, which has favoured the creation of different ways to visualize ontological models depending on the purpose and the meaning of the underlying knowledge.

The TopBraid Composer ontology development tool [28] supports visualization of knowledge graphs in a notation similar to UML (Unified Modelling Language) class diagrams. Analogously, Protégé [29] has several plugins and compatible applications to visualize different aspects of ontologies as graphs: Ontoviz [30], OWLGrEd [31] and VOWL [32] for complex concepts; Ontograf [33] and GrOWL [34] for interoperable representations; NavigOWL for automatic graph layout [35]; OWLViz [36] and Jambalaya [37] for taxonomies; and OWLPropViz [38] for object properties.

Visualization tools for exploration and analysis of large linked data are focused on browsing data registers, rather than depicting deeper semantics. Some challenges of Linked Data visualization are context adaptation, users and data heterogeneity, supporting different tasks (query, combination, filtering, etc.), and more recently, performance [39]. Gephi, a general-purpose graph visualization tool [40], supports data retrieval from several sources, calculation of graph measurements, and automatic graph layout. Similar graph-oriented tools are Payola [41], which enables a complete ecosystem of visualization plugins; LDVizWiz [42], aimed to support non-specialists consuming linked open data; and OntoTrix [43], based on vertex adjacency matrices. Specifically targeted at the Life Sciences domain, Cytoscape can connect to SPARQL endpoints and create graphs [44], and PrEVIEw can visualize data retrieved from the Life Sciences Linked Open Data cloud [45].

Comparative analyses of visualization tools, including ontology modelling and knowledge graph exploration software, yield the conclusion that there is no one-size-fits-all solution, and the selection of tools will depend on the purpose [46]. Existing visualization benchmarks are more focused on coverage than on performance [47,48], and detailed evaluations are scarce. It has not been emphasized until recently that the existing approaches, including those mentioned above, have very limited capability to manage large amounts of data [49], since they do not aim at addressing issues related to performance and scalability; e.g. memory overflow, use of external storage, or efficient automatic layout.

A proposal to address these problems is graphVizdb [50]. This platform supports management and visualization of large linked data graphs by creating a spatial partition of the vertex set, which is afterwards used to allow incremental navigation at different granularity levels. Memo Graph [51] follows a different approach: it extracts relevant descriptors from the graph to generate a summarized model, thus avoiding dealing with the whole structure. Nevertheless, both frameworks are developed in a non-distributed fashion, which results in quite high execution times. In the next section, we present a processing pipeline for graph visualization and a distributed architecture based on Big Data technologies that improves these proposals.

3 Knowledge graph visualization process

3.1 Visualization pipeline

We can distinguish five stages in the generation of graph-based visualizations from linked data: data retrieval, graph building, calculation of metrics, layout and rendering. This workflow is consistent with the Linked Data Visualization Model proposed in [52], which describes the visualization process of RDF and linked data in a generic way. These stages may not be always present or may be executed iteratively.

1. **Data retrieval**: The first step to create a linked data graph is retrieving data. In the Semantic Web context, this will be usually carried out by issuing a SELECT SPARQL query to a triplestore or a federated linked database. The result of this query is a set of triples describing the entities and relationships of interest.

2. **Graph building**: Afterwards, a graph is built from the retrieved data, which can be in-memory or stored in files as plain text (vertices and edge pairs), triples, etc. The objective of this task is to encode the graph in machine-processable format supported by a specific-purpose graph processing library; e.g. Apache TinkerPop [53] or Jung [54] for sequential implementations, or GraphX [55] or GraphFrames [56] for distributed implementations.

3. **Graph calculations:** Different graph measures can be calculated on the graph to improve the visualization. For example, the size and the colour of the vertices can be adjusted according to their degree; by using a more sophisticated ranking procedure, like the PageRank algorithm [57]; or from a combination of different relevance measures [18]. Similarly, other calculations can be performed on the graph edges to determine their features (weight, appearance, etc.), and then applied to define their visual aspect.

4. **Graph layout:** Graph layout is the calculation of the spatial position of the vertices to offer a pleasant depiction and to facilitate the interpretation of the underlying information by the human users. A classic family of graph layout algorithms is force-based vertex placement, which computes attractive and repulsive forces for pairs of vertices to determine how close they should be drawn. Within this family, we can find the Fruchterman-Reingold, the Kamada and Kawai, and the Yifan Hu algorithms [58].

5. **Rendering:** Finally, the graph is displayed on the screen. Usually, a visualization library starts a layered process to translate the higher-level drawing primitives into instructions that are sent to the graphics driver and executed by the graphics card connected to the screen.

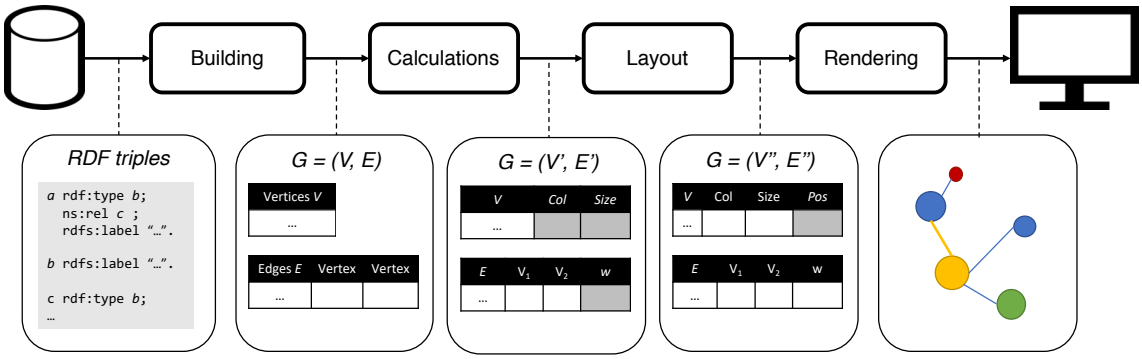


Figure 1 Graph visualization pipeline. A graph *G = (V, E)* is created in the building stage. Vertices and edges are extended with the results of the calculations and with the position computed by the layout algorithm. Finally, the graph is sent to the screen to be displayed.

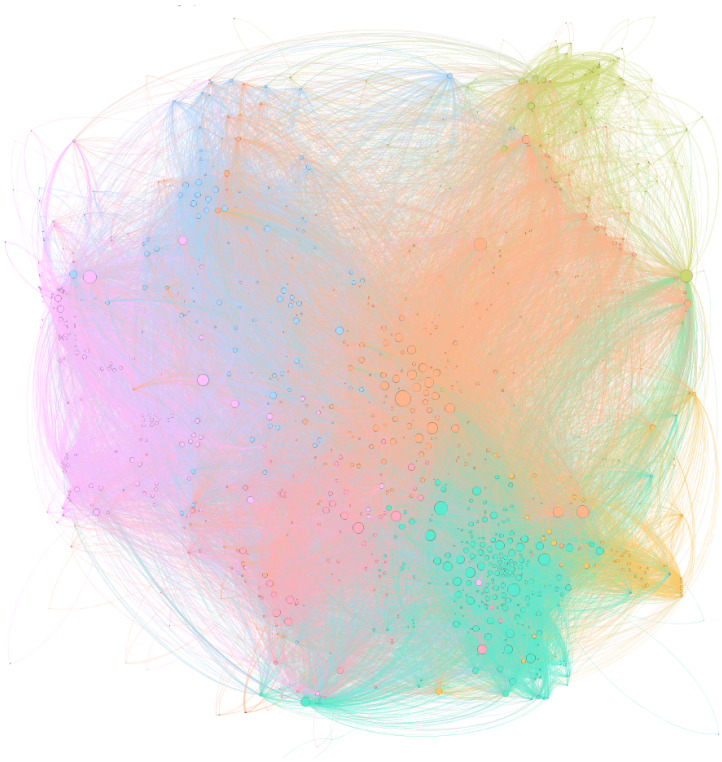


Figure 2 DrugBank drug to drug interactions. Colors are assigned to communities.

The graph building stage does not assume any particular procedure to create graphs from raw data or triples. To support this task, we can use GraphDL [59], an OWL ontology that allows describing graphs by means of a simple vocabulary denoting vertices, edges, and their properties. GraphDL simplifies graph building by allowing the explicit representation of vertices and edges and by providing parsers from GraphDL representations to other formats; e.g. TinkerPop, GraphML and GraphX.

Figure 2 shows an example of the application of the graph visualization pipeline. The graph represents drug to drug interactions in DrugBank [60], encompassing 1,038 vertices (drugs) and 22,927 edges (interactions). Data has been retrieved from the public Bio2RDF SPARQL endpoint[1] and processed with Gephi. Colours are assigned to communities detected by the Louvain method [61], and layout is done with the Fruchterman-Reingold algorithm. The results corresponds to the findings in [62], which discovered 9 functional drug categories from the analysis and visualization of the drug interaction graph.

3.2 Big Data architecture

The increasing size of datasets makes it necessary to implement the visualization pipeline in such a way that it can process large-scale graphs. Big Data frameworks provide support for managing massive amounts of data in reasonable time by using clusters of distributed nodes. Similarly, parallel algorithms, enabling concurrent processing of graph sections, allow us to efficiently deal with large graphs. In this work we have focused on the first type of approaches, but improvements and extensions based on the second ones could be integrated into the proposed architecture [63].

Big Data frameworks include two key elements: (1) a distributed file system; and (2) a distributed computation paradigm. The Apache Hadoop framework is a family of software that implement

[1] http://bio2rdf.org/sparql

both [64]: HDFS (Hadoop Distributed File System), a fault-resilient file system for storage; and MapReduce, an abstract paradigm to implement distributed computation. Hadoop was designed under the principle that "moving computation is cheaper than moving data". Accordingly, it provides a uniform access to the data while facilitating the division of the computing tasks and the gathering of the results across a cluster. The Apache Spark framework complements Hadoop by offering both pure in-memory computation and more flexible ways to divide the computation beyond the relatively simple MapReduce pattern [22,65,66]. Spark includes GraphX, a library for graph management over HDFS which also provides graph processing algorithms [21,55]. Job scheduling in Spark is usually performed by YARN [67], which receives a Spark program, builds a directed acyclic graph of Spark jobs, and manages the cluster resources to run them.

Figure 3 depicts an architecture materializing the visualization pipeline in a Big Data setup. The architecture implements the graph data processing stages in the Hadoop / Spark / GraphX stack. Note that GraphX could be replaced by another graph processing library; e.g. GraphFrames. The triplestore could also be distributed, which is supported by tools such as Virtuoso[2], Blazegraph[3] and Stardog[4], although we do not explore these capabilities in this work.

The RDF Engine retrieves triples describing the knowledge graph from the triplestore. The Graph Builder translate these triples, coming from HDFS or directly from the RDF query Engine, into a graph structure. Different measures can be calculated for the graph, such as PageRank and connected components. The information resulting from these operations can be added to the graph and used in later computations, including layout. Layout is developed as a separated component, because usually it is not directly supported by graph libraries. (A straightforward implementation of the Fruchterman-Reingold layout algorithm in Spark is presented in the next section.) The Renderer delivers the drawing to the available screens (one or several). This stage can be sequential or distributed since it does not require complex calculations nor having the whole graph in memory. A detailed study of this component remains as future research work.

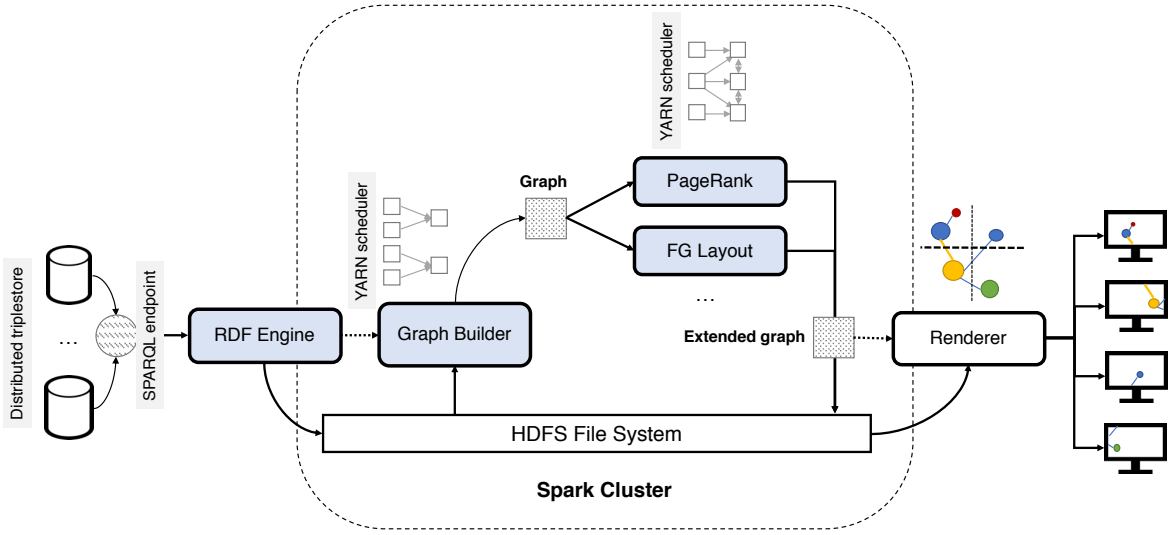


Figure 3 Distributed architecture implementing the graph visualization pipeline in Spark.

[2] https://virtuoso.openlinksw.com/
[3] http://www.blazegraph.com
[4] http://stardog.com

3.3 Implementation

To implement the architecture in Figure 3, we have used Apache Spark as the general engine for large-scale data processing and GraphX for graph management. We have selected GraphX because it is already a part of the Spark framework. The programming language of choice has been Scala, which is recommended for high-performance computation with Spark [66]. The RDF query Engine is implemented with the Eclipse RDF4J[5] framework –formerly known as Sesame–, a widely-used RDF API. The software is publicly available in our repository[6].

The Graph Builder applies a map-reduce strategy to create a GraphX graph from different sources (plain text or triples). In the *map* stage, incoming triples are distributed over the cluster nodes, which identify the kind of information conveyed by the triples —i.e. it is a vertex, an edge, a property, a property value, etc.—and tag them accordingly. In the *reduce* stage, triples describing the same element of the graph —a vertex or an edge— are parsed together along their property value and inserted into a GraphX data structure.

GraphX natively includes implementations for several graph processing algorithms; e.g. PageRank, connected components, label propagation, SVD++, strongly connected components and triangle count. We have selected PageRank for the experiments in this paper because it is a widely known algorithm which exhibits a typical graph processing pattern: running time linearly increases with respect to the number of edges, and the relative amount of computation and communication is constant across iterations [68]. Replacing PageRank and/or adding other algorithms to the implementation is straightforward.

The FG Layout module is a distributed implementation of the Fruchterman-Reingold force-directed algorithm [58] based on GraphX (Figure 4). This algorithm starts with a random placement of the vertices. Next, it calculates repulsive forces for each pair of vertices and attractive forces for each edge, which are used to obtain the updated positions of the vertices. This process is repeated for a fixed number of iterations or until the vertex positions are not anymore updated. Our implementation follows the same steps, adapted to Spark:

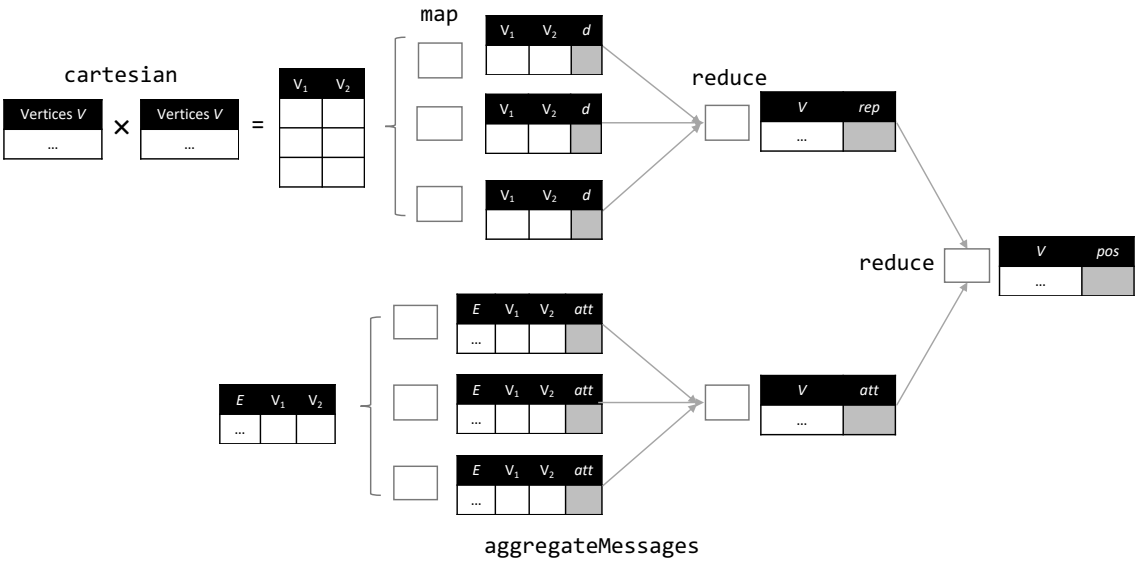


Figure 4 Spark implementation of the Fruchterman-Reingold algorithm.

[5] http://rdf4j.org
[6] https://github.com/jgromero/graphviz-bench

1. First, we calculate the full vertex distance matrix using map-reduce: at the beginning of each iteration, we create a matrix M of size $|V|^2$ storing the distance between each pair of vertices. The distance values are reduced to obtain the repulsive forces.

2. Afterwards, we calculate the attractive forces from the edge list. We use the GraphX instruction `aggregateMessages`, which applies a user-defined function to each edge (i.e. calculation of the attraction force) and then merges the results at the destination vertex.

3. Finally, attractive and repulsive forces are combined to obtain the updated vertex positions.

Optimized extensions of the algorithm could be considered. For instance, GiLA [69] creates a custom partition of the edge list based on the graph properties to better balance the processing load, instead of using the default partitioning strategy of the distributed file system.

4 Experiments

The experiments presented in this section aim at characterizing the stages of the graph visualization process in the Big Data setup. Specifically, we focus on the RDF Engine, the Graph Builder, the PageRank calculation, and the Fruchterman-Reingold Layout. We run a variant of the implementation described in Section 3.3 to measure execution time for different graph types and sizes. As explained above, this implementation used Spark and GraphX for the graph representation, PageRank calculation, and layout. It can transparently work with directed and undirected graphs.

For comparison purposes, a sequential implementation using TinkerPop, Jung and Gephi for graph processing was also developed. Apache TinkerPop is an open source graph computing framework written in Java/Groovy [53] and can be seen as a non-distributed version of GraphX. Jung (the Java Universal Network/Graph Framework) is a Java library that provides optimized implementations of algorithms from graph theory, data mining and social network analysis [54]; e.g. PageRank. Gephi offers a Java library of graph layout algorithms that we can use programmatically [40], including the Fruchterman-Reingold algorithm. These tools are easily interoperable and have shown good performance in previous benchmarks [23,70,71].

First, we compare the results of the distributed and the sequential implementation of the processing pipeline with the Drugbank dataset, concluding that the sequential implementation is not able to deal even with relatively small graphs. Afterwards, we run more experiments with the distributed implementation and the full version of DBPedia, from which we extracted three meaningful graphs with millions of vertices and edges. The results are promising, but the topology of these graphs does not allow us to generalize them. Therefore, we conducted two additional tests with: (1) synthetically-generated graphs with higher density values; (2) a selection of real-world graphs from the Stanford Network Analysis Project (SNAP) repository.

The experiments have been carried out on our Spark cluster, on a cluster of six machines: one driver and five executors. All nodes are identical and each one has 3 TB of disk (Seagate, RAID 1 @ 7200RPM), 100 GB of RAM (Micron @ 1600 MHz), 24 cores (Intel Xeon, E5-2620 v2 @ 2.10 GHz) and a network speed of 10 GB/s. The cluster itself relies on Cloudera CDH 5.13.1 (Spark 2.2.0). We have used 100 executor threads (max.) with 1 core each, allowing up to 20 GB for the driver and 6 GB for the executors.

The sequential version was executed in an iMac equipped with an i7 4 GHz processor, 24 GB of RAM and a 512 GB of SSD disk, and running macOS High Sierra and Java 8.

In the remainder of this section, we will note $|V|$ as the number of vertices; $|E|$ as the number of edges; and D the measure for graph density calculated as:

$$\text{Undirected graphs: } D_U = \frac{2\times|E|}{|V|\times(|V|-1)} \qquad \text{Directed graphs: } D_D = \frac{|E|}{|V|\times(|V|-1)}$$

4.1 DrugBank

In this test, we extracted several random subgraphs representing drug-to-drug interactions from the DrugBank dataset, each one with different $|V|$ value. Isolated vertices were removed from the final subgraph. Extracted graphs were represented with the GraphDL ontology. The SPARQL query to obtain the subgraphs is shown in Listing B1. Execution times with the sequential and the distributed implementations are detailed in Table A1 and Table A2, respectively.

The data source was the linked data version of DrugBank created by the Bio2RDF initiative [72], loaded into an Amazon Web Service (AWS) machine running a Virtuoso triplestore. The data server had the configuration recommended by the creators of Virtuoso image[7]: c3.large instance equipped with 32 GB of SSD disk, 2 CPUs, 3.75 GB of RAM, and 1 Mbps (max.) bandwidth.

Figure 5 shows the times of five executions of the sequential implementation for different $|V|$ and $|E|$. It can be seen that the layout time increases abruptly from 900 vertices (~10,000 edges), making this implementation unusable even for marginally large and dense graphs. In contrast, the times of the data retrieval stage –including network latency– and the graph building stage are relatively stable.

We can compare these sequential execution times with the distributed implementations of the PageRank and the Fruchterman-Reingold algorithms. In Figure 6(a) we can see that the sequential layout takes more than 1 minute to process the complete graph; in contrast, the Spark implementation of the layout process is significantly faster and remains almost constant for the DrugBank subgraphs. Figure 6(b) shows that PageRank calculations are fast in both cases, although the time spent by the sequential implementation increases linearly with size of the graph.

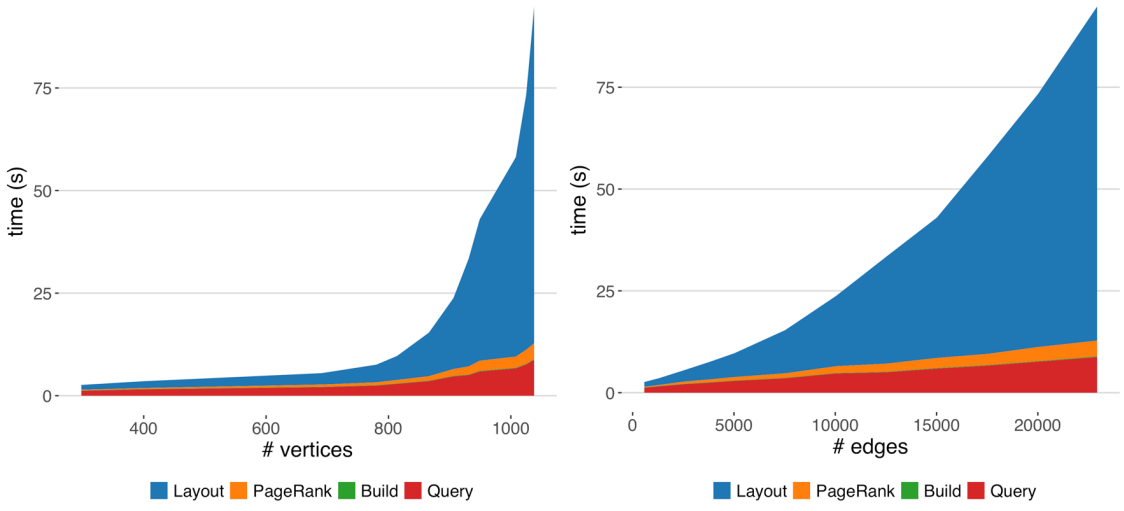
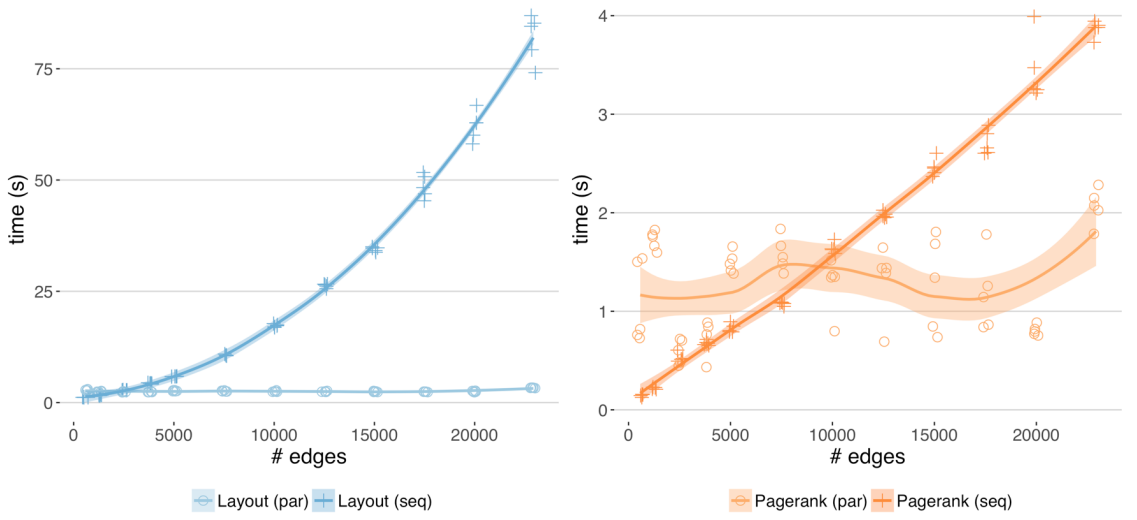


Figure 5 Processing time of the sequential implementation: (a) compared to the number of vertices in the subgraph; (b) compared to the number of edges in the subgraph.

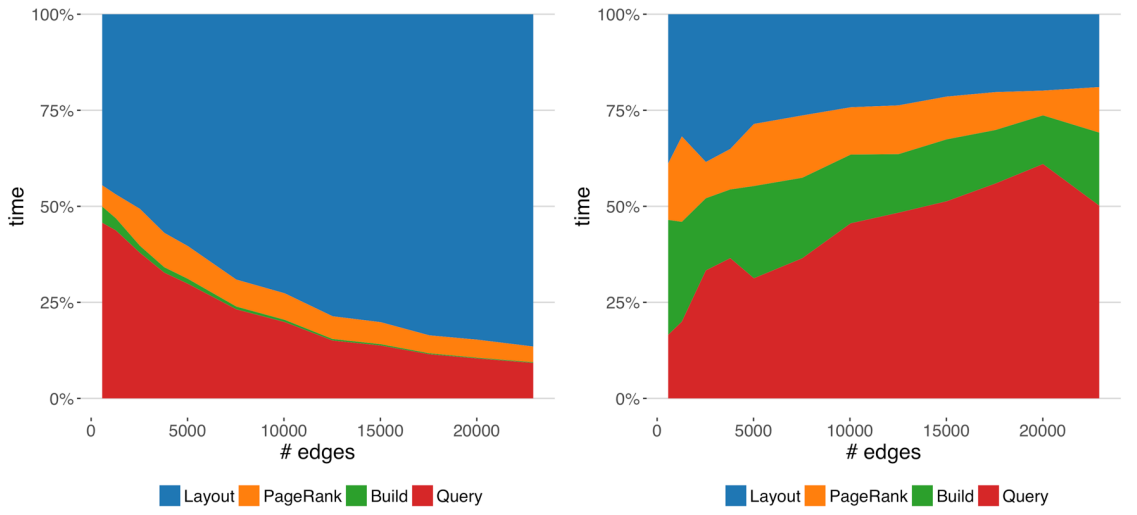
[7] https://aws.amazon.com/marketplace/pp/B00TQ43PS0?qid=1521800621991



(a) (b)

Figure 6 Comparison between processing times of the sequential and the distributed implementations for: (a) Fruchterman-Reingold layout; (b) PageRank calculation. The solid line represents the LOESS regression of each measure. (Note that the scale of the vertical axis is different for (a) and (b).)

Figure 7 compares for different subgraph sizes –in terms of $|E|$– the share of the total processing time spent in each stage by each implementation. In the sequential case, query is quite time-consuming for smaller graphs, but it scales better than the other stages; conversely, graph layout clearly becomes the most demanding task as graphs get more complex. In the distributed case, the percentages of time spent in graph building, PageRank calculation and layout are relatively higher for small graphs, due to the overhead of distributing the calculations; however, query becomes the most demanding task for larger and denser graphs.



(a) (b)

Figure 7 Processing time (in % of total time) of each stage in: (a) the sequential implementation; (b) the distributed implementation.

4.2 DBPedia

In these experiments, we have extracted three knowledge graphs from DBPedia [73], namely:

(1) Artist to artist relations, representing influences between artists
(2) Person to person relations, representing connections between relevant persons.
(3) Page to page relations, representing links between Wikipedia pages.

Table A3 includes detailed information about size and density of these graphs.

Listings B2–4 describe the SPARQL queries to obtain the graphs. Execution times are also detailed in Table A3. In this case, the query stage was carried out offline, producing three RDF files that were stored in the HDFS filesystem. These files were afterwards loaded and parsed to build each graph by a modified version of the Graph Builder –described in Section 3.3– able to read from HDFS instead of from the triplestore. This was purposely done to evaluate graph loading from the filesystem instead of the triplestore performance and the network latency. Accordingly, the *Load* stage subsumes the previous *Query* and *Build*.

The data source was the DBPedia 2016-04 image curated by Ontology2[8], loaded in an Amazon Web Service (AWS) machine running a Virtuoso triplestore. This edition includes more than 1 billion facts, a 168% increase over the public DBPedia SPARQL endpoint[9]. The data server had the configuration recommended by Ontology2: r3.xlarge instance equipped with 80 GB of SSD disk, 4 CPUs and 30.5 GB of RAM.

Figure 8 shows the average processing time for each graph. The performance of the tasks is consistent with the observations of the previous section, in which we saw that the PageRank calculation and the layout stages tend to proportionally have similar performance. The density of the person-to-person and the page-to-page graphs is very low, which makes the $|E|$ value –and consequently, the processing time– not to increase too much despite the x10 increase in $|V|$ for each graph.

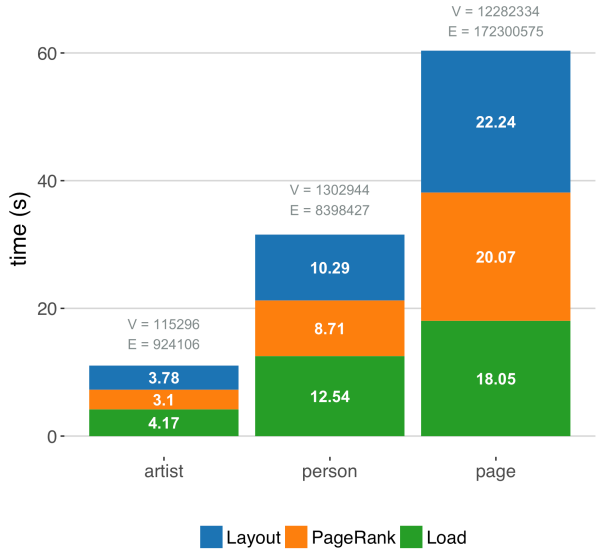
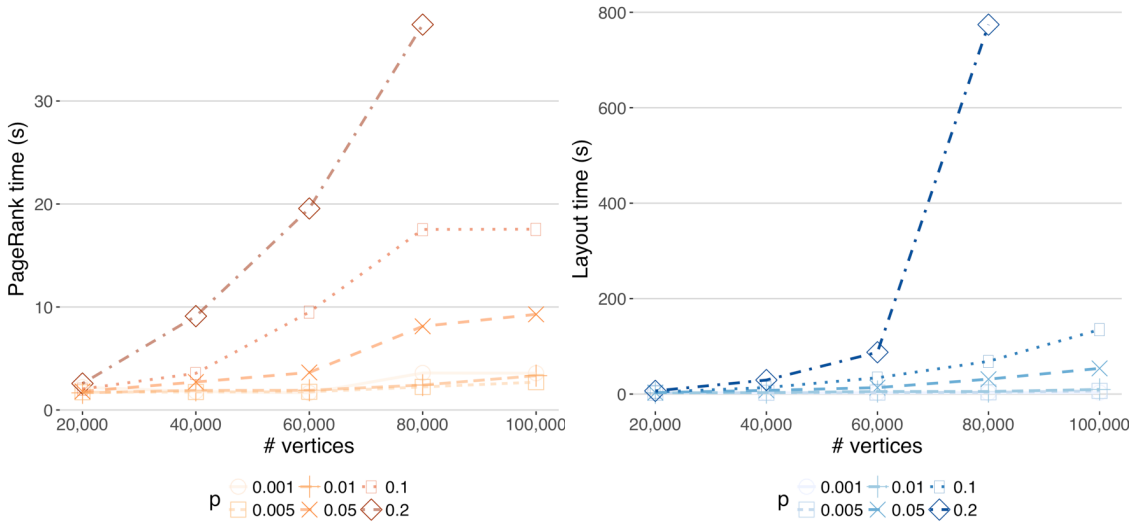


Figure 8 Processing time of DBPedia graphs in the distributed implementation by stage. From left to right: artist influences network, person relations network, Wikipedia page links network.

[8] https://aws.amazon.com/marketplace/pp/B01HMUNH4Q?qid=1521803868351

[9] http://dbpedia.org/sparql



(a) (b)

Figure 9 Processing time of synthetic graphs in the distributed implementation: (a) PageRank; (b) layout. *p* are the probabilities used to generate each set of graphs according to the Erdös-Rényi model and is equal to D_D. (Note that the scale of the vertical axis is different for (a) and (b).)

4.3 Synthetic graphs

To continue the performance evaluation of our distributed implementation, we generated several random graphs with different density values by using the Erdös-Rényi model [74]. This model defines p as the probability of generating an edge (a, b) between two random vertices of the graph. Accordingly, the expected number of edges in a directed graph is $|V| \times |V-1| \times p$, and the expected density is $D_D = p$. The graph generation procedure was implemented with Spark and GraphX.

Table A4 shows the performance of the processing stages in our Spark cluster. The time to generate the graphs is negligible, and the PageRank time is in the same range as in the previous experiments. In Figure 9, we can also see that the layout time does not increase very much for relatively sparse graphs. However, it notably increases for denser graphs with $p >= 0.1$, specially from the 60,000 vertices threshold. For 80,000 vertices and $p = 0.2$, the layout time is above 10 minutes.

Figure 10 depicts the total processing time of the synthetic graphs with $p >= 0.1$ by representing the number of vertices vs the number of edges. The 1 billion edges threshold approximately delimits the frontier between moderate and large execution times.

4.4 SNAP graphs

The Erdös-Rényi model has some limitations for modelling real-world problems [75], since the resulting graphs have low clustering (unlike many social networks), and the vertex degree distribution does not have long tails (as it happens in semantic networks, protein to protein interactions or computer networks, to name some). Therefore, we evaluated the distributed implementation with a selection of assorted graphs with heterogeneous sizes and topologies from the Stanford Large Network Dataset, a publicly available set of social and information real-world networks distributed with the Stanford Network Analysis Platform (SNAP) [76].

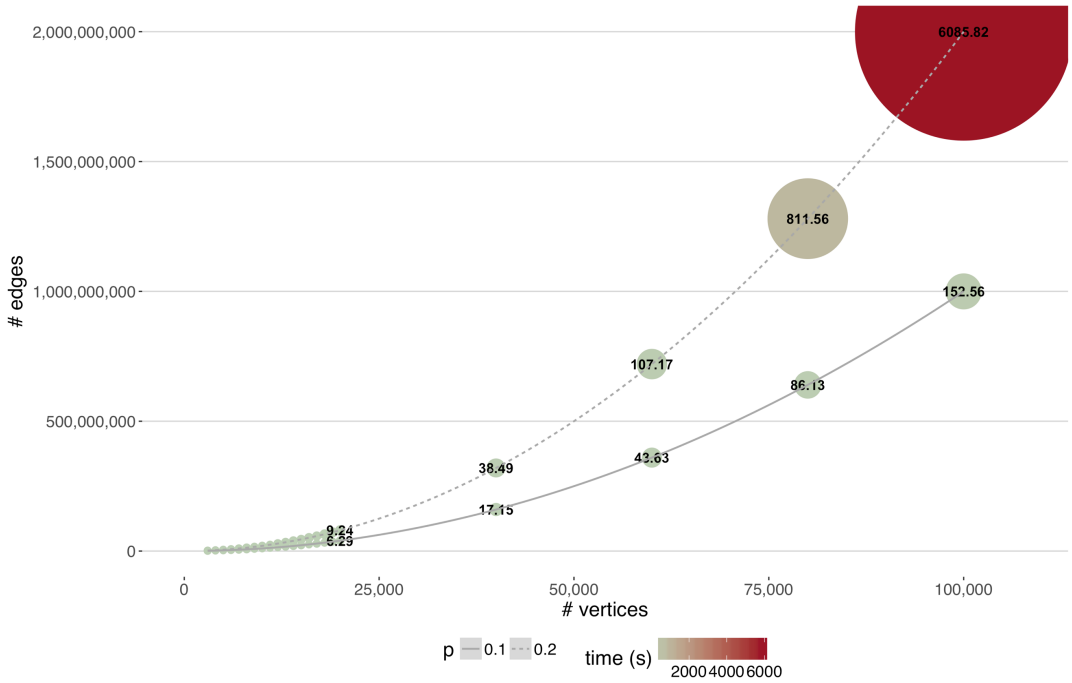


Figure 10 Processing time of synthetic graphs: PageRank + layout. Times are represented with bubble areas and colors. *p* are the probabilities used to generate each set of graphs according to the Erdös-Rényi model and is equal to D_D.

Table A5 shows the performance of the processing stages in our Spark cluster. In this case we also stored the graphs as files in HDFS, which were loaded to build the GraphX graphs. Note that in several cases load time is higher than processing time (Figure 11).

Figure 12 shows that two main graph types can be identified among larger graphs, according to their density values D_D: ≈10^{-5} and ≈10^{-6}. Since these density values are small compared to those used in Section 4.3, execution time does not rise –except for the very large com-Friendster network. As a matter of fact, there is a linear correlation between $(|V|, |E|)$ and the PageRank time for datasets with more than 1 million vertices and density around 10^{-5} (R^2 = 0.9481).

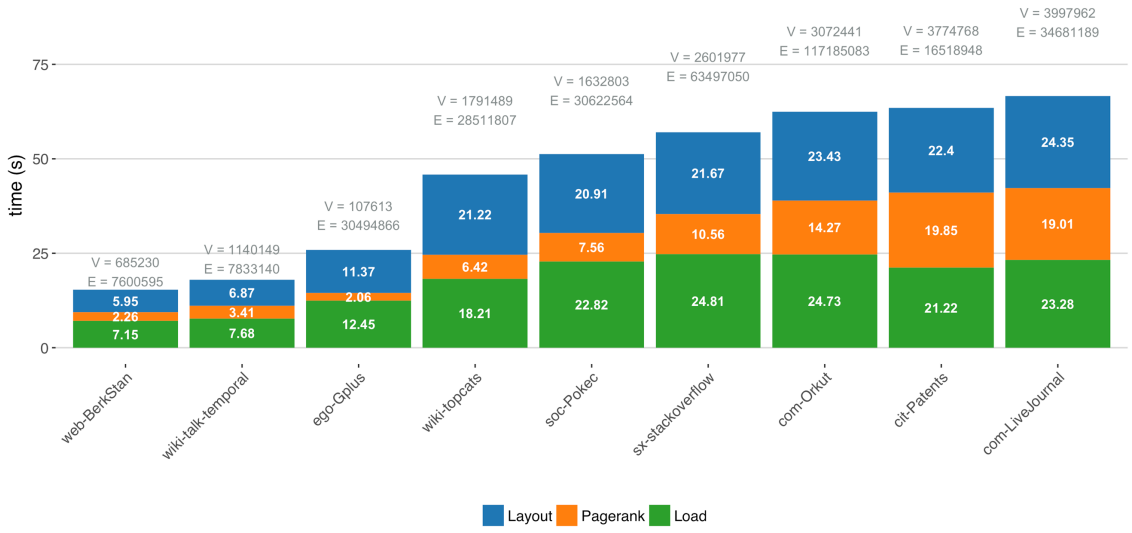


Figure 11 Processing time of SNAP graphs (load time > 5, com-Friendster is excluded).

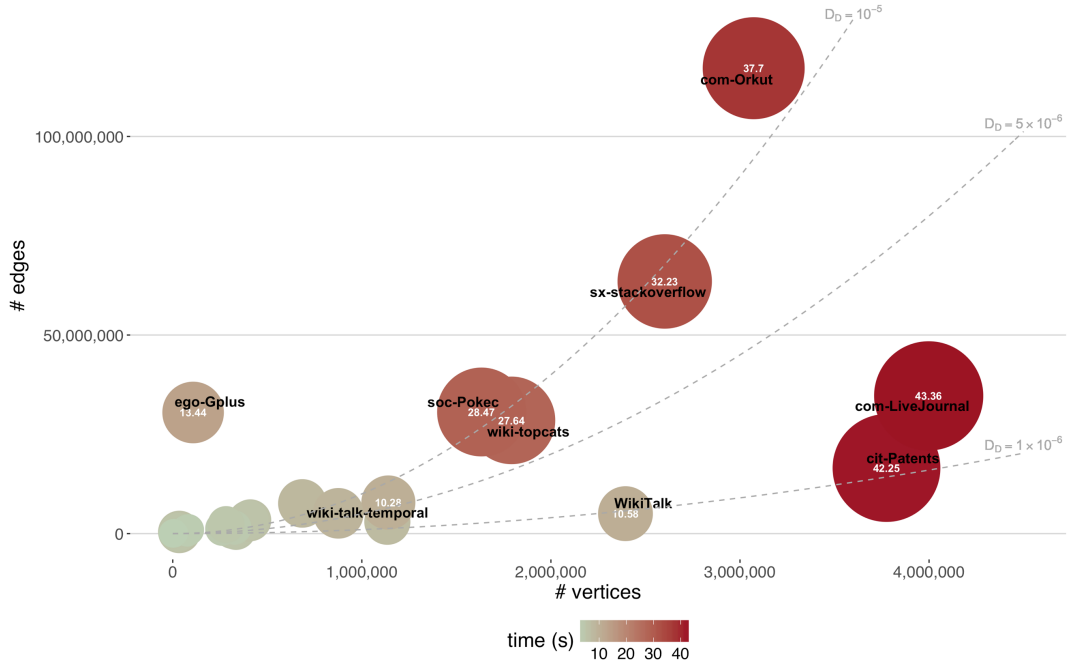


Figure 12 Processing time of SNAP graphs (com-Friendster is excluded): PageRank + layout. Times are represented with bubble areas and colors. Theoretical densities D_D are shown with dashed lines.

5 Discussion

As expected, the distributed implementation based on Spark and GraphX outperformed the sequential implementation even with the relatively small DrugBank dataset. What is more interesting is that we tested that our distributed layout algorithm scales well, yielding execution times comparable to the PageRank included in GraphX (Figure 7(b)). Further optimizations of the distributed layout algorithm (e.g. data partitions, see below) and the cluster configuration (e.g. higher-capability nodes vs more nodes) could be considered. Nevertheless, the first priority should be to improve the triplestore configuration to reduce query time. It would also be convenient to study in more detail the impact of the maximum number of iterations parameter of the Fruchterman-Reingold algorithm.

The distributed implementation also behaved well with DBPedia and SNAP graphs, excluding the very large com-Friendster graph. We can see that the difference between PageRank and layout times is significant for some specific SNAP graphs (Figure 11); e.g. ego-Gplus, soc-Pokec and wiki-topcats. These are large graphs, but still in the region [$|V| < 2{,}000{,}000$, $|E| < 50{,}000{,}000$]. We can conclude that the performance of PageRank is generally better for large real-world (sparser) graphs near the limits of this range, but otherwise (above or below this threshold), in practice it is similar to the performance of the layout process. For small and dense synthetic graphs, PageRank was only slightly more efficient (Figure 9). The differences were however very obvious for larger and denser graphs with $|V| > 60{,}000$. Still, it was possible to process a graph with more than 1 billion vertices in less than 3 minutes (synthetic graph with $p = 0.1$, $|V| = 100{,}000$).

In general, we can say that satisfactory layout times were achieved –in terms of time required to create the visualizations (cf. Gephi API with large graphs) and compared to the native implementations of algorithms such as PageRank–, even though: (a) we did not create any custom partition of the graph to distribute the computation according to the graph topology; (b) we did not reduce the neighbourhood of each vertex to optimize the computation of the distance matrix. For instance, execution times below 25 seconds were achieved for each processing stage

of the DBPedia page-to-page network, which has more than 12 million vertices and 172 million edges. Moreover, the execution time of the layout stage was similar to the GraphX native PageRank function applied to this graph. Furthermore, we were able to work with the com-Friendster network (>65 million vertices and >1.8 billion edges). Our implementation was also faster than the reported times of graphVizdb and GiLA, although the results should not be directly compared due to different overall purpose, programming languages and hardware configurations.

Therefore, we can conclude that very large graph structures, extracted from real-world problems, can be processed without the need of spatial partitions. This is not an argument against such research approaches, which can be necessary if interactive graph exploration at different granularity levels is addressed (as in graphVizdb and Memo Graph). This would be the case of big dense graphs, such as the largest one tested in this work –synthetic graph with 100,000 vertices and more than 2 billion edges. In such scenario, the huge number of edges would make ineffective the layout algorithm, and it would be useless to render the whole graph on the screen.

6 Conclusions and future work

Data visualization plays a prominent role in data exploration and has only increased its importance with the advent of Big Data. This paper has studied the performance of a framework based on Apache Spark and GraphX for visualization of large-scale knowledge graphs. We have seen that large graphs with millions of vertices and edges can be processed at a low cost with state-of-the-art Big Data technologies to generate compelling visualizations. The results are consistent through several experiments over real-world and synthetic datasets with heterogeneous graph topologies.

Our proposal addresses the performance problems of large graph data exploration. Therefore, it is not intended for visualizing complex semantics, for which existing ontology development environments like TopBraid or Protégé are more suited. In contrast, it significantly improves the current state of the art of non-distributed exploration tools such as Gephi and Cytoscape. In the last two years, other proposals aimed at large graph visualization have been presented, such as graphvizDB and GiLA. Although these works focus on interactive and multi-granular visualizations, and in principle present worse performance numbers, it would be interesting to perform a more detailed comparison of the advantages and the drawbacks of each approach, in order to identify the cases in which spatial partitions and levels are necessary and to evaluate further improvements (shared variables, message-passing, heuristic edge and vertex sets cuts, etc.). To do so, specific-purpose profiling tools such as GiViP could be used [77]. New graph management libraries for Spark, such as GraphFrames, should also be considered, as well are more specific and diverse topologies [76].

We plan to continue our research work by studying the usability of the resulting graph visualizations (cf. Figure 13(d)). To do so, we will leverage the capabilities of the Data Observatory (DO)[10], a large-scale visualization environment built at the Data Science Institute at Imperial College London. The DO features a distributed data processing and rendering cluster connected to 64 46" high definition screens that enable a 313º immersive space with a total resolution of 132M pixels across 37.31m^2. The DO will help us to investigate how people work and interact with large graph visualizations, how they can be improved for visual analytics, and which capabilities should be optimized in the graph-processing software. We will also further develop the GraphDL ontology and the associated toolbox, since they offer a useful framework to describe, produce and transform graphs in a flexible way by using a common format.

[10] http://www.imperial.ac.uk/data-science/data-observatory-/

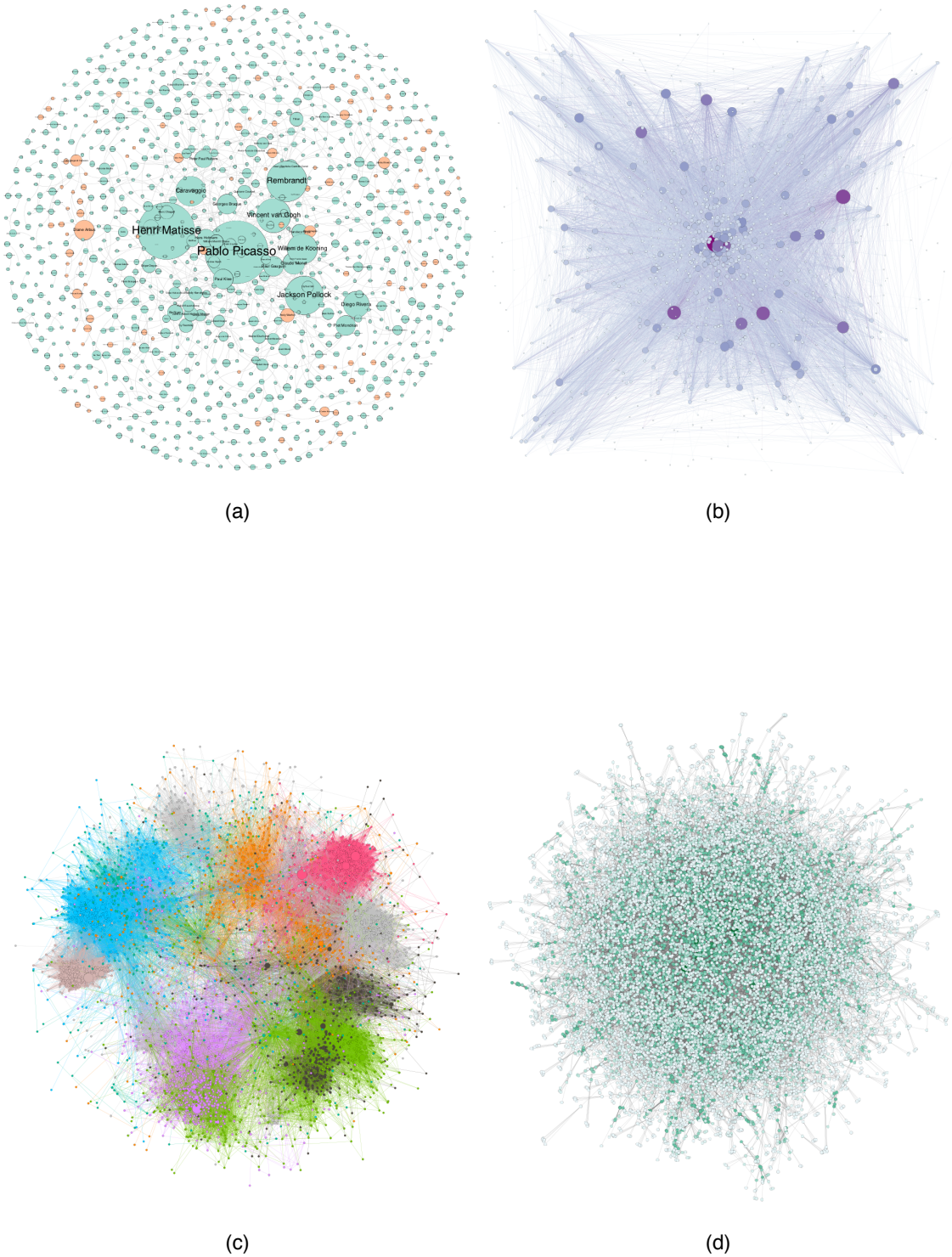


Figure 13 Excerpts of result graphs rendered with Gephi: (a) artist influences network, filtered by most relevant painters and photographers (vertex size is proportional to degree); (b) SNAP email-Eu-core (vertex size and color is proportional to PageRank); (c) SNAP ego-Facebook (vertex size is proportional to PageRank, color is assigned to Louvain communities) (d) SNAP com-DBLP.

Acknowledgements

This work was supported by the Spanish Government (TIN2015-64776-C3-1-R project). Juan Gómez-Romero is supported by University of Granada under the Young Researchers Fellowship, and the Spanish Ministry of Education, Culture and Sport under the José Castillejo Research Stays Programme. Miguel Molina-Solana is supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 743623. The authors gratefully acknowledge technical and financial support from the Data Science Institute at Imperial College London, which provided the computational infrastructure used in this research work, and from the members of the Visualization group, who helped to polish the charts.

Bibliography

[1] B.H. McCormick, Visualization in scientific computing, ACM SIGBIO Newsl. 10 (1988) 15–21. doi:10.1145/43965.43966.

[2] J. Leigh, A. Johnson, L. Renambot, T. Peterka, B. Jeong, D.J. Sandin, J. Talandis, R. Jagodic, S. Nam, H. Hur, Y. Sun, Scalable resolution display walls, Proc. IEEE. 101 (2013) 115–129. doi:10.1109/JPROC.2012.2191609.

[3] U.M. Fayyad, G.G. Grinstein, A. Wierse, Information visualization in Data Mining and Knowledge Discovery, MK/Morgan Kaufmann Publishers, San Francisco, CA (USA), 2002.

[4] D.A. Keim, Visual exploration of large data sets, Commun. ACM. 44 (2001) 38–44. doi:10.1145/381641.381656.

[5] B. Shneiderman, The eyes have it: a task by data type taxonomy for information visualizations, in: Proc. 1996 IEEE Symp. Vis. Lang., IEEE Comput. Soc. Press, 1996: pp. 336–343. doi:10.1109/VL.1996.545307.

[6] P. Ristoski, H. Paulheim, Semantic Web in data mining and knowledge discovery: A comprehensive survey, J. Web Semant. 36 (2016) 1–22. doi:10.1016/j.websem.2016.01.001.

[7] K. Janowicz, F. Van Harmelen, J.A. Hendler, P. Hitzler, Why the data train needs semantic rails, Artif. Intell. 36 (2015) 5–14. doi:http://dx.doi.org/10.1609/aimag.v36i1.2560.

[8] G. Schreiber, Y. Raimond, RDF 1.1 Primer, (2014). https://www.w3.org/TR/rdf11-primer/ (accessed February 19, 2018).

[9] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph, OWL2 Web Ontology Language Primer (Second Edition), (2012). https://www.w3.org/TR/owl2-primer/ (accessed February 19, 2018).

[10] D. Newman, T. Baldwin, L. Cavedon, E. Huang, S. Karimi, D. Martinez, F. Scholer, J. Zobel, Visualizing search results and document collections using topic maps, J. Web Semant. 8 (2010) 169–175. doi:10.1016/j.websem.2010.03.005.

[11] C. Nikolaou, K. Dogani, K. Bereta, G. Garbis, M. Karpathiotakis, K. Kyzirakos, M. Koubarakis, Sextant: Visualizing time-evolving linked geospatial data, J. Web Semant. 35 (2015) 35–52. doi:10.1016/j.websem.2015.09.004.

[12] M. Martin, K. Abicht, C. Stadler, A.-C. Ngonga Ngomo, T. Soru, S. Auer, CubeViz – Exploration and visualization of statistical Linked Data, in: Proc. 24th Int. Conf. World Wide Web (WWW 2015), 2015: pp. 219–222. doi:10.1145/2740908.2742848.

[13] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, R. V. Kenyon, J.C. Hart, The CAVE: audio visual experience automatic virtual environment, Commun. ACM. 35 (1992) 64–72. doi:10.1145/129888.129892.

[14] A. Febretti, A. Nishimoto, T. Thigpen, J. Talandis, L. Long, J.D. Pirtle, T. Peterka, A. Verlo,

M. Brown, D. Plepys, D. Sandin, L. Renambot, A. Johnson, J. Leigh, CAVE2: a hybrid reality environment for immersive simulation and information analysis, in: Proc. IS&T / SPIE Electron. Imaging, Eng. Real. Virtual Real., 2013. doi:10.1117/12.2005484.

[15] G. Wallace, O.J. Anshus, Peng Bi, Han Chen, Yuqun Chen, D. Clark, P. Cook, A. Finkelstein, T. Funkhouser, A. Gupta, M. Hibbs, Kai Li, Zhiyan Liu, R. Samanta, R. Sukthankar, O. Troyanskaya, Tools and applications for large-scale display walls, IEEE Comput. Graph. Appl. 25 (2005) 24–33. doi:10.1109/MCG.2005.89.

[16] K. Doerr, F. Kuester, CGLX: A scalable, high-performance visualization framework for networked display environments, IEEE Trans. Vis. Comput. Graph. 17 (2011) 320–332. doi:10.1109/TVCG.2010.59.

[17] D. McGinn, D. Birch, D. Akroyd, M. Molina-Solana, Y. Guo, W.J. Knottenbelt, Visualizing dynamic Bitcoin transaction patterns, Big Data. 4 (2016) 109–119. doi:10.1089/big.2015.0056.

[18] M. Molina-Solana, D. Birch, Y. Guo, Improving data exploration in graphs with fuzzy logic and large-scale visualisation, Appl. Soft Comput. 53 (2017) 227–235. doi:10.1016/j.asoc.2016.12.044.

[19] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, D.W. Fellner, Visual analysis of large graphs: State-of-the-art and future research challenges, Comput. Graph. Forum. 30 (2011) 1719–1749. doi:10.1111/j.1467-8659.2011.01898.x.

[20] R. Pienta, J. Abello, M. Kahng, D.H. Chau, Scalable graph exploration and visualization: Sensemaking challenges and opportunities, in: Proc. IEEE Int. Conf. Big Data Smart Comput. (BIGCOMP 2015), 2015: pp. 271–278. doi:10.1109/35021BIGCOMP.2015.7072812.

[21] R.S. Xin, J.E. Gonzalez, M.J. Franklin, I. Stoica, GraphX: A resilient distributed graph system on Spark, in: First Int. Work. Graph Data Manag. Exp. Syst., New York, New York, USA, 2013. doi:10.1145/2484425.2484427.

[22] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: Proc. 2nd USENIX Conf. Hot Top. Cloud Comput., 2010. doi:10.1007/s00256-009-0861-0.

[23] F. Holzschuher, R. Peinl, Querying a graph database – language selection and performance considerations, J. Comput. Syst. Sci. 82 (2016) 45–68. doi:10.1016/J.JCSS.2015.06.006.

[24] S.A. Khan, M.A. Qadir, M.A. Abbas, M.T. Afzal, OWL2 benchmarking for the evaluation of knowledge based systems, PLoS One. 12 (2017). doi:10.1371/journal.pone.0179578.

[25] B. Fu, N.F. Noy, M.-A. Storey, Eye tracking the user experience – An evaluation of ontology visualization techniques, Semant. Web. 8 (2016) 23–41. doi:10.3233/SW-140163.

[26] J.F. Sowa, Conceptual Graphs, in: F. van Harmelen, B. Porter, V. Lifschitz (Eds.), Handb. Knowl. Represent., Elsevier B.V., 2008: pp. 213–237. doi:http://dx.doi.org/10.1016/S1574-6526(07)03005-2.

[27] F. Baader, D.L. Mcguiness, D. Nardi, P. Patel-Schneider, The Description Logic Handbook: Theory, implementation and applications, 2nd Ed., Cambridge University Press, 2010.

[28] TopQuadrant, TopBraid Platform Overview, (2018). https://www.topquadrant.com/technology/topbraid-platform-overview/ (accessed February 19, 2018).

[29] M. Musen, The Protégé Project: A look back and a look forward, AI Matters. 1 (2015) 4–12. doi:10.1126/science.1249098.Sleep.

[30] M. Sintek, OntoViz, (2007). https://protegewiki.stanford.edu/wiki/OntoViz (accessed

March 26, 2018).

[31] L. Renārs, M. Grasmanis, U. Bojārs, OWLGrEd ontology visualizer, in: Proc. ISWC-DEV, 2014: pp. 37–42.

[32] S. Lohmann, S. Negru, F. Haag, T. Ertl, Visualizing ontologies with VOWL, Semant. Web. 7 (2016) 399–419. doi:10.3233/SW-150200.

[33] S. Falconer, OntoGraf, (2016). https://protegewiki.stanford.edu/wiki/OntoGraf (accessed March 26, 2018).

[34] S. Krivov, R. Williams, F. Villa, GrOWL: A tool for visualization and editing of OWL ontologies, Web Semant. Sci. Serv. Agents World Wide Web. 5 (2007) 54–57. doi:10.1016/J.WEBSEM.2007.03.005.

[35] A. Hussain, K. Latif, A.T. Rextin, A. Hayat, M. Alam, Scalable visualization of semantic nets using power-law graphs, Appl. Math. Inf. Sci. 8 (2014) 355–367. doi:10.12785/amis/080145.

[36] M. Horridge, OWLViz, (2010). https://protegewiki.stanford.edu/wiki/OWLViz (accessed March 26, 2018).

[37] M.-A. Storey, M. Musen, J. Silva, C. Best, R. Fergerson, N. Ernst, Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé, in: Proc. 7th Int. Conf. Intell. User Interfaces (IUI '02), 2002: p. 239. doi:10.1145/502716.502778.

[38] L. Wachsmann, OWLPropViz, (2008). https://protegewiki.stanford.edu/wiki/OWLPropViz (accessed March 26, 2018).

[39] A.-S. Dadzie, M. Rowe, Approaches to visualising Linked Data: A Survey, Semant. Web. 2 (2011) 89–124. doi:10.3233/SW-2011-0037.

[40] M. Bastian, S. Heymann, M. Jacomy, Gephi: An open source software for exploring and manipulating networks, in: 3rd Int. AAAI Conf. Weblogs Soc. Media, 2009: pp. 361–362. doi:10.1136/qshc.2004.010033.

[41] J. Klímek, J. Helmich, M. Nečaský, Payola: Collaborative linked data analysis and visualization framework, in: Proc. 10th Ext. Semant. Web Conf. (ESWC 2013), Springer, Berlin, Heidelberg, 2013: pp. 147–151. doi:10.1007/978-3-642-41242-4_14.

[42] G.A. Atemezing, R. Troncy, Towards a linked-data based visualization wizard, in: Proc. 5th Int. Conf. Consum. Linked Data, 2014: pp. 1–12.

[43] B. Bach, E. Pietriga, I. Liccardi, Visualizing Populated Ontologies with OntoTrix, Int. J. Semant. Web Inf. Syst. 9 (2013) 17–40. doi:10.4018/ijswis.2013100102.

[44] M.E. Smoot, K. Ono, J. Ruscheinski, P.L. Wang, T. Ideker, Cytoscape 2.8: New features for data integration and network visualization, Bioinformatics. 27 (2011) 431–432. doi:10.1093/bioinformatics/btq675.

[45] S. Sana, Q. Mehmood, D. Zehra, PrEVIEw : Clustering and visualising PubMed using visual interface, in: Proc. 2nd Int. Work. Vis. Interact. Ontol. Linked Data, 2016: pp. 17–27.

[46] M. Dudáš, O. Zamazal, V. Svátek, Roadmapping and navigating in the ontology visualization landscape, in: Proc. Int. Conf. Knowl. Eng. Knowl. Manag. (EKAW 2014), 2014: pp. 137–152. doi:10.1007/978-3-319-13704-9_11.

[47] F. Haag, S. Lohmann, S. Negru, T. Ertl, OntoViBe 2: Advancing the ontology visualization benchmark, in: P. Lambrix, E. Hyvönen, E. Blomqvist, V. Presutti, G. Qi, U. Sattler, Y. Ding, C. Ghidini (Eds.), Proc. EKAW 2014 Satell. Events, Springer International Publishing, Cham, 2015: pp. 83–98. doi:10.1007/978-3-319-17966-7_9.

[48] HOBBIT Project, Benchmark IV – Visualisation & Services, (2016). https://project-hobbit.eu/outcomes/benchmark-iv-visualisation-services/ (accessed March 26, 2018).

[49] N. Bikakis, T. Sellis, Exploration and visualization in the web of Big Linked Data: A survey of the state of the art, in: Proc. EDBT/ICDT 2016 Jt. Conf., 2016. http://ceur-ws.org/Vol-1558/paper28.pdf (accessed March 26, 2018).

[50] N. Bikakis, J. Liagouris, M. Krommyda, G. Papastefanatos, T. Sellis, graphVizdb: A scalable platform for interactive large graph visualization, in: IEEE 32nd Int. Conf. Data Eng. (ICDE 2016), IEEE, 2016: pp. 1342–1345. doi:10.1109/ICDE.2016.7498340.

[51] F. Ghorbel, F. Hamdi, N. Ellouze, E. Métais, F. Gargouri, Visualizing large-scale Linked Data with Memo Graph, Procedia Comput. Sci. 112 (2017) 854–863. doi:10.1016/J.PROCS.2017.08.079.

[52] J.M. Brunetti, S. Auer, R. García, J. Klímek, M. Nečaský, Formal Linked Data visualization model, in: Proc. Int. Conf. Inf. Integr. Web-Based Appl. Serv., ACM Press, New York, New York, USA, 2013: pp. 309–318. doi:10.1145/2539150.2539162.

[53] M.A. Rodriguez, The Gremlin graph traversal machine and language, Proc. 15th ACM Symp. Database Program. Lang. (DBLP 2015). (2015) 1–10. doi:10.1145/2815072.2815073.

[54] J. O'Madadhain, D. Fisher, S. White, Y.-B. Boey, The JUNG (Java Universal Network/Graph) Framework, (2016). https://github.com/jrtom/jung.

[55] M.S. Malak, R. East, Spark GraphX in action, Manning, 2016.

[56] A. Dave, A. Jindal, L.E. Li, R. Xin, J. Gonzalez, M. Zaharia, GraphFrames, in: Proc. Fourth Int. Work. Graph Data Manag. Exp. Syst. (GRADES '16), ACM Press, New York, New York, USA, 2016: pp. 1–8. doi:10.1145/2960414.2960416.

[57] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: Bringing order to the web, Stanford InfoLab, 1999.

[58] S.G. Kobourov, Force-directed drawing algorithms, in: R. Tamassia (Ed.), Handb. Graph Draw. Vis., CRC Press, 2016: pp. 383–408.

[59] J. Gómez-Romero, M. Molina-Solana, The GraphDL ontology, (2017). https://github.com/jgromero/graphdl.

[60] D.S. Wishart, Y.D. Feunang, A.C. Guo, E.J. Lo, A. Marcu, J.R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda, N. Assempour, I. Iynkkaran, Y. Liu, A. Maciejewski, N. Gale, A. Wilson, L. Chin, R. Cummings, D. Le, A. Pon, C. Knox, M. Wilson, DrugBank 5.0: A major update to the DrugBank database for 2018, Nucleic Acids Res. 46 (2018) D1074–D1082. doi:10.1093/nar/gkx1037.

[61] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. (2008). doi:10.1088/1742-5468/2008/10/P10008.

[62] L. Udrescu, L. Sbârcea, A. Topîrceanu, A. Iovanovici, L. Kurunczi, P. Bogdan, M. Udrescu, Clustering drug-drug interaction networks with energy model layouts: community analysis and drug repurposing, Sci. Rep. 6 (2016) 32745. doi:10.1038/srep32745.

[63] D.S. Banerjee, A. Kumar, M. Chaitanya, S. Sharma, K. Kothapalli, Work efficient parallel algorithms for large graph exploration on emerging heterogeneous architectures, J. Parallel Distrib. Comput. 76 (2015) 81–93. doi:10.1016/J.JPDC.2014.11.006.

[64] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop Distributed File System, in: Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST 2010), 2010: pp. 1–10. doi:10.1109/MSST.2010.5496972.

[65] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, Learning Spark, O'Reilly, 2015.

[66] T. Kajdanowicz, P. Kazienko, W. Indyk, Parallel processing of large graphs, Futur. Gener. Comput. Syst. 32 (2014) 324–337. doi:10.1016/J.FUTURE.2013.08.007.

[67] V.K. Vavilapalli, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J.

Lowe, H. Shah, Apache Hadoop YARN, in: Proc. 4th Annu. Symp. Cloud Comput., 2013: pp. 1–16. doi:10.1145/2523616.2523633.

[68] J.E. Gonzalez, R.S. Xin, A. Dave, D. Crankshaw, M.J. Franklin, I. Stoica, GraphX: Graph processing in a distributed dataflow framework, in: Proc. 11th USENIX Conf. Oper. Syst. Des. Implement., 2014: pp. 599–613.

[69] A. Arleo, W. Didimo, G. Liotta, F. Montecchiani, Large graph visualizations using a distributed computing platform, Inf. Sci. (Ny). 381 (2017) 124–141. doi:10.1016/J.INS.2016.11.012.

[70] M. Jacomy, T. Venturini, S. Heymann, M. Bastian, ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software, PLoS One. 9 (2014) 1–12. doi:10.1371/journal.pone.0098679.

[71] L. Wang, A graphics library for system analysis, Technical University of Denmark, 2012.

[72] A. Callahan, J. Cruz-Toledo, M. Dumontier, Ontology-Based Querying with Bio2RDF's Linked Open Data., J. Biomed. Semantics. 4 (2013) S1. doi:10.1186/2041-1480-4-S1-S1.

[73] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia, Semant. Web. 6 (2015) 167–195. doi:10.3233/SW-140134.

[74] P. Erdös, A. Rényi, On random graphs, Publ. Math. 6 (1959) 290–297.

[75] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Rev. Mod. Phys. 74 (2002) 47–97. doi:10.1103/RevModPhys.74.47.

[76] J. Leskovec, R. Sosič, SNAP: A general purpose network analysis and graph mining Library, ACM Trans. Intell. Syst. Technol. 8 (2016) 1–20. doi:10.1145/2898361.

[77] A. Arleo, W. Didimo, G. Liotta, F. Montecchiani, GiViP: A visual profiler for distributed graph processing systems, in: Proc. Int. Symp. Graph Draw. Netw. Vis. (GD 2017), 2017: pp. 256–271. doi:10.1007/978-3-319-73915-1_21.

Appendix A: Data Tables

$\|V\|$	$\|E\|$	D_U	*Query*	*Build*	*PageRank*	*Layout*
298	574	0.013	1.20	0.11	0.15	1.17
400	1283	0.016	1.55	0.11	0.22	1.66
691	2528	0.011	2.09	0.10	0.53	2.79
780	3795	0.012	2.48	0.11	0.68	4.31
866	7530	0.020	3.56	0.11	1.08	10.62
906	10,039	0.024	4.72	0.14	1.64	17.27
931	12,522	0.029	5.03	0.13	1.98	26.27
1,008	17,539	0.035	6.67	0.15	2.71	48.59
1,025	20,010	0.038	7.62	0.15	3.44	62.15
1,038	22,927	0.043	8.74	0.19	3.87	82.02

Table A1 Performance of processing stages with the DrugBank dataset, Java sequential implementation: average time of 5 executions (s).

$\|V\|$	$\|E\|$	D_U	*Query*	*Build*	*PageRank*	*Layout*
298	574	0.013	1.20	2.18	1.07	2.83
400	1283	0.016	1.55	2.02	1.72	2.47
691	2528	0.011	2.09	1.18	0.59	2.41
780	3795	0.012	2.48	1.22	0.72	2.38
866	7530	0.020	3.56	2.05	1.58	2.57
906	10,039	0.024	4.72	1.85	1.28	2.51
931	12,522	0.029	5.03	1.59	1.32	2.47
1,008	17,539	0.035	6.67	1.66	1.18	2.42
1,025	20,010	0.038	7.62	1.58	0.80	2.49
1,038	22,927	0.043	8.74	2.29	2.06	3.31

Table A2 Performance of processing stages with the DrugBank dataset, Spark distributed implementation: average time of 5 executions (s). Query times are the same for the sequential and the distributed implementations.

	$\|V\|$	$\|E\|$	D_D	*Load*	*PageRank*	*Layout*
Artist	115,296	924,106	0.000070	4.17	3.10	3.78
Person	1,302,944	8,398,427	0.000005	12.54	8.71	10.29
Page	12,282,334	172,300,575	0.000001	18.05	20.07	22.24

Table A3 Performance of processing stages with the DBpedia dataset, Spark implementation: average time of 5 executions (s). Data load is performed from the serialized graphs represented in GraphDL and stored in HDFS.

P	$\|V\|$	$\|E\|$	D_D	*Generation*	*PageRank*	*Layout*
0.001	4,456	26,544	0.001337	0.01	2.32	2.45
0.001	9,888	103,920	0.001063	0.01	1.68	2.40
0.001	40,000	1,596,320	0.000998	0.02	1.79	2.53
0.001	80,000	6,386,120	0.000998	0.02	3.59	2.76
0.001	100,000	9,997,104	0.001000	0.03	3.57	3.16
0.005	5,000	122,608	0.004905	0.01	2.06	2.45
0.005	10,000	494,704	0.004948	0.01	1.82	2.40
0.005	40,000	7,986,400	0.004992	0.01	1.74	2.70
0.005	80,000	31,947,160	0.004992	0.02	2.24	3.64
0.005	100,000	49,982,720	0.004998	0.02	2.70	6.61
0.010	5,000	247,616	0.009907	0.01	2.10	2.48
0.010	10,000	992,384	0.009925	0.01	1.61	2.44
0.010	40,000	16,031,032	0.010020	0.01	1.91	3.08
0.010	80,000	64,001,520	0.010000	0.02	2.42	5.35
0.010	100,000	99,995,904	0.010000	0.02	3.33	9.69
0.050	5,000	1,246,648	0.049876	0.01	2.20	2.56
0.050	10,000	4,967,992	0.049685	0.01	1.66	2.59
0.050	40,000	79,919,968	0.049951	0.01	2.71	7.71
0.050	80,000	319,953,120	0.049993	0.02	8.12	31.24
0.050	100,000	499,840,304	0.049985	0.02	9.29	53.82
0.100	5,000	2,506,680	0.100287	0.01	2.15	2.66
0.100	10,000	9,983,880	0.099849	0.01	1.83	2.92
0.100	40,000	159,895,864	0.099937	0.01	3.56	13.59
0.100	80,000	639,839,824	0.099976	0.02	17.53	68.60
0.100	100,000	999,883,800	0.099989	0.02	17.55	135.00
0.200	5,000	5,001,928	0.200117	0.01	2.32	2.92
0.200	10,000	19,988,880	0.199909	0.01	2.04	3.64
0.200	40,000	320,008,888	0.200011	0.01	9.11	29.38
0.200	80,000	1,280,039,920	0.200009	0.02	37.42	774.14
0.200	100,000	2,000,077,856	0.200010	0.02	222.96	5,862.86

Table A4 Performance of processing stages with synthetic datasets, Spark implementation: average time of 5 executions (s).

		$\|V\|$	$\|E\|$	D	*Load*	*PageRank*	*Layout*
cit-HepPh	D	34,546	421,578	0.000353	3.66	2.64	3.58
cit-HepTh	D	27,770	352,807	0.000458	2.63	2.12	3.23
cit-Patents	D	3,774,768	16,518,948	0.000001	21.22	19.85	22.40
com-Amazon	U	3,34,863	925,872	0.000017	2.16	2.27	3.21
com-DBLP	U	317,080	1,049,866	0.000021	1.94	2.07	3.15
com-Friendster	U	65,608,366	1.81E+09	0.000001	134.36	385.26	340.83

com-LiveJournal	U	3,997,962	34,681,189	0.000004	23.28	19.01	24.35
com-Orkut	D	3,072,441	1.17E+08	0.000025	24.73	14.27	23.43
com-Youtube	U	1,134,890	2,987,624	0.000005	4.04	2.87	4.53
email-Eu-core	D	1,005	25,571	0.025342	0.13	0.27	2.41
wiki-topcats	D	1,791,489	28,511,807	0.000009	18.21	6.42	21.22
amazon0302	D	262,111	1,234,877	0.000018	1.11	0.95	3.02
amazon0312	D	400,727	3,200,440	0.000020	2.64	1.56	4.30
amazon0505	D	410,236	3,356,824	0.000020	2.64	1.67	4.36
amazon0601	D	403,394	3,387,388	0.000021	2.60	1.59	3.94
ego-Facebook	U	4,039	88,234	0.010820	0.15	0.26	2.44
ego-Gplus	D	107,613	30,494,866	0.002633	12.45	2.06	11.37
soc-Epinions1	D	75,879	508,837	0.000088	0.49	0.45	2.56
soc-Pokec	D	1,632,803	30,622,564	0.000011	22.82	7.56	20.91
soc-sign-bitcoinalpha	D	3,783	24,186	0.001690	0.11	0.24	2.38
soc-sign-bitcoinotc	D	5,881	35,592	0.001029	0.11	0.28	2.35
soc-Slashdot0811	D	77,360	905,468	0.000151	0.85	0.52	2.74
soc-Slashdot0902	D	82,168	948,464	0.000140	0.74	0.56	2.75
wiki-Vote	D	7,115	103,689	0.002049	0.15	0.26	2.41
sx-stackoverflow	D	2,601,977	63,497,050	0.000009	24.81	10.56	21.67
wiki-talk-temporal	D	1,140,149	7,833,140	0.000006	7.68	3.41	6.87
web-BerkStan	D	685,230	7,600,595	0.000016	7.15	2.26	5.95
web-Google	D	875,713	5,105,039	0.000007	4.53	3.16	5.79
web-NotreDame	D	325,729	1,497,134	0.000014	1.25	0.98	3.03
web-Stanford	D	281,903	2,312,497	0.000029	2.05	1.17	3.38
Wiki-Vote	D	7,115	103,689	0.002049	0.15	0.25	2.37
WikiTalk	D	2,394,385	5021,410	0.000001	4.39	5.24	5.34

Table A5 Performance of processing stages with SNAP datasets, Spark implementation: average time of 5 executions (s). Data load is performed from plain text files in HDFS. Graphs are classified into SNAP categories: citation networks, networks with ground truth, product co-purchasing, social networks, temporal graphs, web graphs, Wikipedia graphs.

Appendix B: SPARQL queries

```
prefix drugbank: <http://bio2rdf.org/drugbank_vocabulary:>
prefix dcterms:  <http://purl.org/dc/terms/>
prefix graphdl:  <http://ugritlab.ugr.es/graphdl#>

construct {
  ?i  a graphdl:Edge ;
        graphdl:source ?d1 ;
        graphdl:target ?d2 .
  ?d1 a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id; graphdl:val ?d1_name ;
                 a graphdl:AttributeValue ] .
  ?d2 a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id; graphdl:val ?d2_name ] ;
```

```
                a graphdl:AttributeValue ] .
} where {
  ?d1 drugbank:ddi-interactor-in ?i ;
      dcterms:title ?d1_name  ;
      a drugbank:Drug .
  ?d2 drugbank:ddi-interactor-in ?i ;
      dcterms:title ?d2_name ;
      a drugbank:Drug .
  filter(str(?d1) != str(?d2)) .
} LIMIT <N>
```

Listing B1 SPARQL query to build DrugBank drug-to-drug interaction subgraphs represented in GraphDL

```
prefix dbo:     <http://dbpedia.org/ontology/>
prefix foaf:    <http://xmlns.com/foaf/0.1/>
prefix dcterms: <http://purl.org/dc/terms/>
prefix graphdl: <http://ugritlab.ugr.es/graphdl#>

construct {
  []  a graphdl:Edge ;
        graphdl:source ?a ;
        graphdl:target ?b .
  ?a  a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id ;   graphdl:val ?a_id ] ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute dcterms:name ; graphdl:val ?a_name ] .
  ?b  a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id ;   graphdl:val ?b_id ] ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute dcterms:name ; graphdl:val ?b_name ] .
} where {
  ?a  a  dbo:Person ;
      foaf:name ?a_name .
  ?a ?p ?b .
  ?b  a  dbo:Person ;
      foaf:name ?b_name .
  bind(str(?a) as ?a_id)
  bind(str(?b) as ?b_id)
}
```

Listing B2 SPARQL query to build the DBPedia artist-to-artist subgraph represented in GraphDL

```
prefix umbel:   <http://umbel.org/umbel/rc/>
prefix foaf:    <http://xmlns.com/foaf/0.1/>
prefix dcterms: <http://purl.org/dc/terms/>
prefix graphdl: <http://ugritlab.ugr.es/graphdl#>

construct {
  []  a graphdl:Edge ;
        graphdl:source ?a ;
        graphdl:target ?b .
  ?a  a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id ;   graphdl:val ?a_id ] ;
```

```
        graphdl:hasAttributeValue
                [graphdl:forAttribute dcterms:name ; graphdl:val ?a_name ] .
  ?b  a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id ;   graphdl:val ?b_id ] ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute dcterms:name ; graphdl:val ?b_name ] .
} where {
  ?a  a  umbel:Artist ;
      foaf:name ?a_name .
  ?a ?p ?b .
  ?b  a  umbel:Artist ;
      foaf:name ?b_name .
  bind(str(?a) as ?a_id)
  bind(str(?b) as ?b_id)
}
```

Listing B3 SPARQL query to build the DBPedia person-to-person subgraph represented in GraphDL

```
prefix dbo:     <http://dbpedia.org/ontology/>
prefix graphdl: <http://ugritlab.ugr.es/graphdl#>

construct {
  []  a graphdl:Edge ;
        graphdl:source ?a ;
        graphdl:target ?b .
  ?a  a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id ;   graphdl:val ?a_id ] .
  ?b  a graphdl:Node ;
        graphdl:hasAttributeValue
                [graphdl:forAttribute graphdl:id ;   graphdl:val ?b_id ] .
} where {
  ?a  dbo:wikiPageWikiLink ?b .
  bind(str(?a) as ?a_id)
  bind(str(?b) as ?b_id)
}
```

Listing B4 SPARQL query to build the DBPedia page-to-page subgraph represented in GraphDL