Datenbank Spektrum
https://doi.org/10.1007/s13222-019-00306-x

SCHWERPUNKTBEITRAG

BIGGR: Bringing Gradoop to Applications

M. Ali Rostami[1] · Matthias Kricke[1] · Eric Peukert[1] · Stefan Kühne[1] · Moritz Wilke[1] · Steffen Dienst[1] · Erhard Rahm[1]

Received: 23 January 2019 / Accepted: 25 January 2019

Abstract
Analyzing large amounts of graph data, e.g., from social networks or bioinformatics, has recently gained much attention. Unfortunately, tool support for handling and analyzing such graph data is still weak and scalability to large data volumes is often limited. We introduce the BIGGR approach providing a novel tool for the user-friendly and efficient analysis and visualization of Big Graph Data on top of the open-source software KNIME and GRADOOP. Users can visually program graph analytics workflows, execute them on top of the distributed processing framework Apache Flink and visualize large graphs within KNIME. For visualization, we apply visualization-driven data reduction techniques by pushing down sampling and layouting to GRADOOP and Apache Flink. We also discuss an initial application of the tool for the analysis of patent citation graphs.

Keywords Graph analysis · Graph visualization · Graph sampling · Gradoop · KNIME

1 Introduction

The analysis of large network and graph datasets is becoming of increasing interest, for example, to gain insights from social networks, protein interaction networks in bioinformatics or in business applications, e.g., in logistics. Graph analytics aims at analyzing such networks, especially complex relationships between heterogeneous data entities of interest. There are many approaches to manage and analyze graph data including graph databases and distributed graph processing platforms, but they still have substantial limitations regarding either scalability to large datasets, support for complex graph mining (e.g., clustering, frequent pattern mining, etc.) or ease of use (flexible graph data model, graph visualization) [1]. For these reasons, we developed the open-source graph analysis platform GRADOOP [2, 3] at ScaDS[1], that aims at overcoming the limitations of previous graph processing platforms.

GRADOOP provides a flexible graph data model based on extended property graphs [5] together with a variety of powerful operators including pattern matching, graph grouping and aggregation [6, 7] as well as a library of graph mining algorithms, e.g., for frequent subgraph mining [8]. The operators and algorithms are implemented on top of Apache Flink and can therefore be executed on shared-nothing clusters to process large amounts of data in parallel.

Programming graph analysis workflows that consist of multiple data transformation and analysis steps is cumbersome and requires high technical expertise from the analyst, which represents a considerable usability hurdle. Moreover, graph data mining tasks are typically embedded in data mining workflows including operators for data preparation, machine learning and result visualization. However, existing tools for constructing such data mining workflows such as

✉ M. Ali Rostami
rostami@informatik.uni-leipzig.de

Matthias Kricke
kricke@informatik.uni-leipzig.de

Eric Peukert
peukert@informatik.uni-leipzig.de

Stefan Kühne
kuehne@informatik.uni-leipzig.de

Moritz Wilke
wilke@informatik.uni-leipzig.de

Steffen Dienst
dienst@informatik.uni-leipzig.de

Erhard Rahm
rahm@informatik.uni-leipzig.de

[1] Institute for Informatics, University of Leipzig, Augustusplatz 10, 04109 Leipzig, Germany

[1] Competence Center for Scalable Data Services and Solutions (ScaDS Dresden/Leipzig) [4].

Published online: 14 February 2019

KNIME [9], Kepler [10], RapidMiner [11] or Galaxy [12] do not yet allow for handling and visualizing Big Graph Data with a huge number of vertices and edges. Therefore, we propose a tool which supports (1) the visual modeling of graph analytics workflows, (2) executes these workflows in a distributed fashion using GRADOOP, and (3) efficiently visualizes Big Graph Data by pushing complex visualization-specific computations down to a distributed server system for parallel execution.

We developed this tool called BIGGR on top of GRADOOP and the data science platform KNIME[2]. In this paper, we describe the approach and its initial application. In particular, we make the following contributions:

- We give an overview to the BIGGR tool for the user-friendly and efficient analysis and visualization of large Big Graph Data. The tool allows users to visually define graph analysis workflows involving GRADOOP operators and existing KNIME operators. We sketch some of the challenges of integrating GRADOOP and Flink into KNIME to achieve a distributed execution that is transparent to the user.
- We introduce the BIGGR approach for visualizing large graphs. It pushes down visualization-specific computations such as layouting and sampling as operators to GRADOOP to be efficiently executed in a distributed fashion.
- We present a real-world application of the BIGGR approach to analyze patent data.

After an initial discussion of related work and an introduction to GRADOOP and KNIME in Sect. 3, we describe how GRADOOP is integrated into KNIME in Sec. 4. In Sect. 5, we dive into our developed tooling for visualizing large graphs in KNIME. Sect. 6 introduces the real-world use case for patent data and Sect. 7 finally gives conclusions and sketches future work.

2 Related Work

There are several workflow-based data science tools that can be used to integrate and orchestrate multiple independently built data mining components or operators for data analysis or data manipulation. These tools are often domain-specific for certain kinds of data (e.g., genomic sequence data in bioinformatics or spectrometry data in chemistry) and typically support a user to visually build workflows in the form of operator trees that can be automatically executed and reused (see [13] for a recent survey and comparison). Unfortunately, only some of them have some initial support for Big Data, namely KNIME, [9], Kepler [10], Apache Taverna [14], RapidMiner [11] and Galaxy [12], by supporting distributed execution, e.g., by providing some integration with Big Data frameworks. For instance, KNIME and RapidMiner offer workflow operators (called nodes) for loading data from and storing to Big Data stores such as HDFS[3], HIVE[4], or IMPALA[5] and also provide an integration with APACHE SPARK[6] that maps workflow nodes to Spark operators. Scientific workflow management tools like Kepler [10], Apache Taverna [14] or Galaxy [12] have a special focus on executing workflows on HPC computing infastructures. Within another ScaDS project, such HPC support was recently also added for KNIME [15].

However, all these tools lack support for analyzing Big Graph Data. Initial attempts to support graph analysis have been made with Galaxy, which is widely used in bioinformatics [12]. It supports a so-called cluster-adapter with the Apache Spark Driver to run Spark jobs and can also utilize Spark's distributed graph processing API GraphX via the so-called GraphFlow [16] or SparkGalaxy [17] front-ends. Since Galaxy does not support graph data, results are not visualized as graphs and the integration is done by storing and retrieving inputs and outputs from and to HDFS. The KNIME tool provides an initial abstraction for graph data, but it has to be processed sequentially so that big graph mining tasks are not yet supported. The BIGGR project aims at improving on this by using the distributed graph analysis system GRADOOP. Furthermore, we aim at providing advanced visualization support for Big Graph Data.

Visualizing Big Graph Data is a well researched topic and many techniques have been proposed to speed up layouting for large graphs [18]. Unfortunately, the problem is still not sufficiently solved as shown in a more recent overview for visualizing linked data graphs [19]. Existing graph databases such as Neo4J[7] or an extension by Oracle[8] offer support for visualizing graph data. However, such tools mostly do not scale well for large graphs and the results are often cluttered as was shown for Neo4J in a DBpedia case study [20]. The authors point out that the dense structure of vertices and edges requires simplification approaches to make the visualizations comprehensible. In our work, we investigate such simplification approaches and in particular follow a so-called visualisation-driven data

[2] The technology transfer into KNIME is funded within a joint BMBF project between ScaDS / University of Leipzig and Knime. It is planned to make the described extensions freely available within an upcoming release of KNIME.

[3] http://hadoop.apache.org.

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

[5] https://impala.apache.org.

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

[7] https://neo4j.com.

[8] https://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytix.

reduction (VDDR) that was recently proposed by Jugel et. al. [21] for relational data. In VDDR, the visualization system pushes down data reduction logic to the data source to reduce workload on a visualization client with the goal of not significantly changing the actual visualization result. To our knowledge, we are the first to apply this concept for graph visualization and we are pushing sampling, preprocessing, and layouting down to Gradoop and Apache Flink to speedup graph visualization for the client.

3 Background

3.1 GRADOOP

GRADOOP is an open source framework for scalable analytics of large-semantically expressive graphs [3]. To achieve horizontal scalability of storage and processing capacity, GRADOOP runs on shared-nothing clusters and utilizes existing open-source frameworks. The difficulties of distributing data and computation are transparent for users who can focus on the problem domain. GRADOOP implements the *Extended Property Graph Model* (EPGM) [5] which describes how graphs and their elements (vertices and edges) are structured. It is an extension of the popular property graph model [22, 23] used in various graph database systems. EPGM supports heterogeneous and schema-free graphs with vertices and edges of different types (labels) and with possibly different properties. The graph data is not limited to a single graph, but there can be collections of possibly overlapping graphs. These graphs can also have a label and properties.

EPGM also defines a set of declarative graph operators on single graphs and graph collections. These operators can be called with a the domain-specific language GRALA (*G*raph *A*nalytical *La*nguage) to implement a program (workflow) for graph analysis. Table 1 shows available graph operators and graph algorithms categorized by their input. Besides general operators for graph transformation or aggregation, GRADOOP also provides *pattern matching* [6] capabilities known from graph database systems and analytical operators, e.g., for *graph grouping* [7] and further structural graph transformations [24]. Moreover, the auxiliary operators *apply* and *call* can be used to seamlessly integrate user-specific operators in the analysis programs.

GRADOOP supports several ways to store graph data. The GRALA interface *DataSource* is used to read and *DataSink* to write graph data. Hence, for each *DataSink* an appropriate *DataSource* exists. GRADOOP supports HDFS file-based graph storage in CSV format or distributed database storage in Apache HBase or Apache Accumulo. GRALA also hides the implementation of the data model and its operators within the underlying distributed execution engine Apache Flink. Apache Flink provides high-level APIs that enable fast application development by abstracting from the complexities of distributed computing. Moreover, Apache Flink provides several libraries that can be combined and integrated within a GRADOOP program, e.g., for graph processing, machine learning, and SQL.

GRADOOP offers a Java API containing the EPGM abstraction including all operators defined within GRALA. This way, users can specify analytical programs and execute them either locally for testing or on a cluster. Graphs can optionally be initialized from existing datasets which allows for pre-processing data within the dataflow system. Graphs also expose the underlying datasets, which enables post-processing using any available library provided by the dataflow system.

We already evaluated GRADOOP and its operators in previous publications using artificial and real-world hetero-

Table 1 Analytical graph operators and algorithms available in GRADOOP organized by their input type

	Unary operators	Binary operators	Graph algorithms
Logical graph	Aggregation	Combination	Page rank
	Pattern Matching [6]	Overlap	Community detection
	Transformation	Exclusion	Connected components
	Grouping [7]	Equality	Single Source Shortest Path
	Subgraph		Summarization
	Call		Hyperlink-Induced Topic Search
	Sampling		Frequent subgraph mining
	Vertex fusion		Graph statistics
Graph collection	Selection	Union	Frequent Subgraph Mining [8]
	Distinct	Intersection	
	Limit	Difference	
	Apply	Equality	
	Reduce		
	Matching		
	Call		

geneous graphs with up to 10 billion edges. In particular, we demonstrated good runtimes and scalability for the Cypher-based pattern matching operator [6] and the graph grouping operator [7]. A benchmark for a comprehensive GRALA program including several GRADOOP operators and a community detection algorithm was conducted in [5] and achieved near-linear scalability. For our implementation of Frequent Subgraph Mining [8], we were also able to show high scalability for growing data volumes and increasing computing resources.

3.2 KNIME

KNIME Analytics Platform [9] is an open-source software for creating data science applications and services. Analysis workflows can be visually created with a drag-and-drop style graphical interface, without the need for coding. A workflow in KNIME corresponds to a directed graph of so-called nodes where each node encapsulates a certain processing functionality for data transformation or analysis. For data input and export, KNIME supports different text formats such as CSV, XML, and JSON as well as unstructured data types like images. There are several connectors from KNIME to databases and cloud platforms as well as access to Twitter, Google Sheets and Azure datasets. There are more than 2000 nodes and hundreds of publicly available workflow examples.

Different visualizations are available in KNIME, from classic ones (bar chart, scatter plot) to advanced charts (parallel coordinates, sunburst, network graph). For example, a summary statistics about columns in a KNIME table can be reported in different formats including PDF.

4 Integration of GRADOOP into KNIME

To integrate GRADOOP into KNIME, we created KNIME nodes for most existing GRADOOP operators, data sources, and data sinks together with an extensive documentation. A complete list of operators is given in Table 1 and their visual representation in the KNIME node repository is shown in the left part of Fig. 1. Within the project, we added some visualization-specific operators to GRADOOP such as for graph sampling that will be discussed in Sect. 5.

The user can drag and drop operators into the workflow editor of KNIME and connect them to build complex graph analytics workflows. When executing the workflow, a GRADOOP operator DAG is built behind the scenes and is executed on a Flink cluster. The analysis results are shown in graph or tabular format. An example of such a workflow is shown in Fig. 1b in which we analyze patent data. We provide more details about this analysis use case in Sect. 6.

For the integration of GRADOOP operators, we had to tackle several technical challenges such as adding a scalable graph visualization to KNIME (Sect. 5) and running Flink jobs out of KNIME. The main challenge was to deal with the lazy evaluation [25] of Flink. In this computational model, a sequence of Flink functions is executed only by either explicitly triggering execution or by requesting output to a data sink (like writing to a file). When appending further operators to an already executed Flink workflow, the complete workflow is executed (and optimized) again from the beginning. This is in contrast to the design of KNIME (and other existing workflow systems such as Galaxy) in which each node can be executed and visualized as soon as the input values are available. Attaching new nodes to

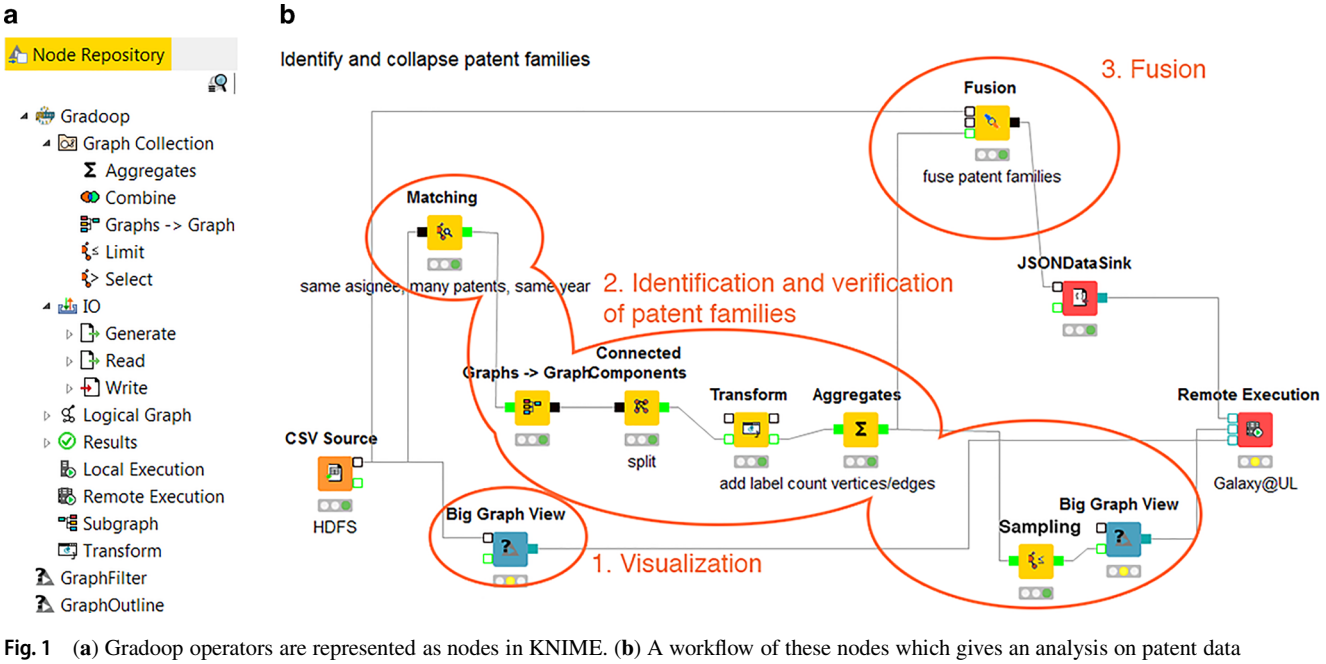

Fig. 1 (**a**) Gradoop operators are represented as nodes in KNIME. (**b**) A workflow of these nodes which gives an analysis on patent data

a KNIME workflow or changing parameters of individual nodes typically does not require to re-execute the complete workflow from the beginning.

To deal with the mismatch between the distributed execution platform Flink and the workflow metaphor of KNIME, we had to introduce a workaround. In particular, we added so-called *executor nodes* to KNIME to explicitly trigger the execution of Flink code either on a local machine (e.g., for small data volumes or simple visualizations) or on a remote cluster. As a result, each workflow that involves Gradoop operators needs to have an executor node which builds the whole Flink workflow for all preceding nodes and executes them. On the other hand, the execution of intermediate nodes (except an executor and graph viewer) does not produce any results in KNIME for the current implementation. For the sample workflow in Fig. 1, the final node *RemoteExecution* on the far right of the figure triggers the workflow execution on a cluster at the University of Leipzig, called Galaxy Cluster.

While the workaround is obviously not the optimal solution, it actually turned out to by quite useful since we could control the execution without requiring to change the KNIME tool and existing KNIME workflows. Also, special data sources and data sinks can be built that connect to existing KNIME nodes. These nodes would need to copy data from and to the cluster to be available for Flink.

Newer versions of Flink already allow a user to partially change Flink operator chains at run-time (e.g., changing parameters of individual functions and the degree of parallelism). By using checkpoints we could also get rid of repeated executions after changing a workflow. Moreover, KNIME is also considering alternative execution models that differ from node-wise execution.

These changes might be opportunities to improve the current implementation.

5 Visualization of big graphs

The visualization of a graph with millions of vertices and edges is often very time-consuming and the result can easily be too cluttered to be useful. We therefore need to reduce the number of vertices and edges for visualization, e.g., by suitable sampling or grouping techniques, while still conveying the most relevant information. In our project, we follow two directions. First, we offer special nodes in KNIME that support the user in analyzing visualization-specific properties of a graph and those help in reducing the size of a graph by sampling or grouping. For that purpose, we also newly developed parallel versions of these operators in GRADOOP that are executed in parallel on top of Flink. Second, we developed an interactive Big Graph Viewer that pushes down complex visualization-specific computations

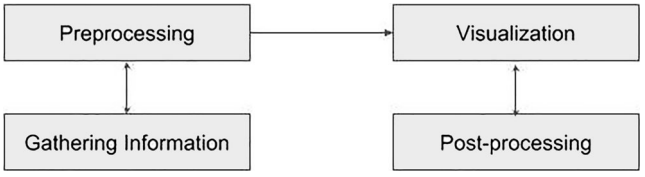

Fig. 2 Steps for interactive visualization of large graphs

such as layouting and sampling to Flink which significantly speeds up visualization in the client. This approach adapts parts of the visualization-driven data reduction (VDDR) approach from Jugel et. al. [21] to graph visualization.

A typical user needs to employ four steps to visualize Big Graph Data that are shown in Fig. 2. Initially a user gathers general information about the graph to later select appropriate sampling and layouting techniques. This involves collecting statistical information about the graph structure, such as the number of vertices, edges, triangles, the degree distribution and so on. This information is determined by a new node called *Graph Statistics*. Further Gradoop nodes such as grouping can be applied to determine semantic information, like value distributions and frequencies of vertex/edge labels and properties. For example, in geographical data, geographical distributions of entities like cities or countries could be computed. With the grouping node, we can also determine a "schema graph", by performing a label-wise aggregation of all vertices and edges [7].

After gathering information, the user can add preprocessing operators (such as sampling or grouping) to the workflow. These operators reduce the number of edges and vertices to be visualized in the *Big Graph View* on the client. The *Big Graph View* initially computes a simple layout and draws the computed graph. The user can now interactively change the sampling and layouting or apply filters to the graph for exploration. We call this step postprocessing. Instead of letting the visualization client compute complex layouts which are often iterative in nature, we generate a GRADOOP workflow in the background and execute layout and sampling operators in parallel on a flink cluster. The resulting graph and computed node positions are returned to the client for drawing. The following subsections discuss preprocessing, visualization, and postprocessing in more detail.

5.1 Preprocessing

There are many known techniques for sampling and graph reduction, but not all of them can easily be implemented in a parallel fashion on top of the Flink framework. In the following, we discuss approaches that we implemented so far.

Initially, we added a GRADOOP operator (and KNIME node) for basic *degree-based filtering* which eliminates dis-

connected or weakly connected vertices by finding the so-called k-core [26, 27] of graphs. For this purpose, we compute connected components and let the user select individual components for visualization. Often, a primary analysis can be done on a small component which can be generalized to other components to a certain extent.

In a second step, we added several graph sampling methods [28] to GRADOOP that select vertices or edges from the complete graph based on a given probability distribution. In particular, we implemented the following methods that can be used within a new KNIME node called Sampling:

- Random vertex sampling: Vertices are filtered if a generated random value is below a given threshold.
- Random edge sampling: Edges are filtered if a generated random value is below a given threshold.
- Vertex edge sampling: Based on a random value, first some vertices are selected. Then, some of their edges are filtered based on another random threshold.
- Vertex neighborhood sampling: After the random selection of some vertices, their neighborhood is randomly filtered based on a given threshold.
- Limited-degree vertex sampling: This sampling is similar to vertex sampling. The difference is that only vertices with degrees below some threshold are filtered. This is helpful especially when vertices with high degrees contain major information.
- Non-uniform vertex sampling: This sampling method is similar to vertex sampling but the random value is generated non-uniformly.
- PageRank sampling: The vertices are filtered based on their PageRank values.

The evaluation of these sampling methods is currently under way.

5.2 Visualization

After preprocessing, the graph layout is computed and the graph is drawn such that the required information is visualized correspondingly. For this purpose, we implemented a web-based visualization which allows to reuse the visualization for other uses of GRADOOP. We make use of two existing Javascript libraries for the graph visualization: Cytoscape.js[9] for smaller graphs and VivaGraph[10] for bigger graphs. The two libraries complement each other, because the first one provides more flexibility for visualization and the latter one can visualize, in our experience, up to ten times bigger graphs.

We further developed layouting techniques for specific types of graphs. In particular, we investigated visualizations for clusters and similarity graphs in entity resolution where vertices in a cluster should be positioned near to each other. These techniques are also published as a separate tool called SIMG-VIZ [29]. Fig. 3 shows a sample visualization where entities of the same cluster have the same color. The links between entities come from the underlying similarity graph and are either correct (green) or wrong (red) w.r.t. the correct match clustering. Multi-color vertices indicate entities assigned to more than one cluster which should be corrected, e.g., by applying a different match configuration or clustering algorithm.

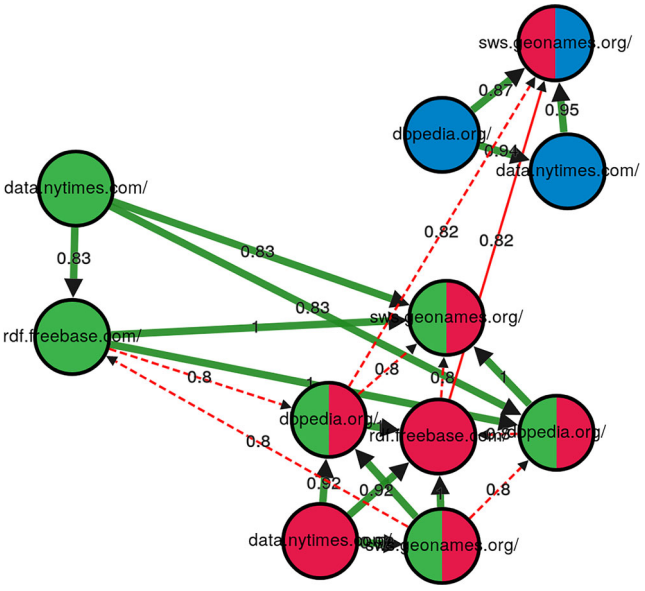

Fig. 3 SIMG-VIZ visualization for clusters of linked entities

5.3 Post-processing

After generating a visualization, users can interactively explore the result, e.g., to zoom into a graph, to color or filter some vertices, or to change the layout. The main challenge in this part is that interactivity requires fast execution time. For this purpose, the viewer pushes down the complex operations to the server for a parallel computation in Gradoop and Flink (Fig. 4). Then, the returned results are visualized in a browser instance (e.g., in Chrome).

Fig. 5 shows the Big Graph View that visualizes some patent data. The visualized graph is already simplified by grouping. It shows patents of the recent years grouped by their international patent code (IPC) and cite relations. It can be observed that the patent groups labeled with codes “H04” (electric communication technique) and “G06” (computing, calculating and counting) are bigger as well as the connecting edges are drawn with stronger lines indicating a high relative share among all patents and cite relations. This means that the considered companies

[9] http://js.cytoscape.org.

[10] https://github.com/anvaka/VivaGraphJS.

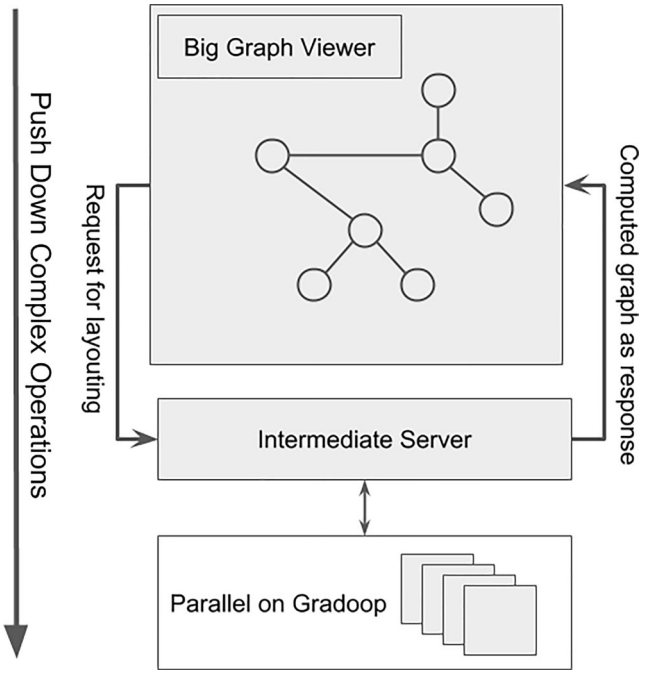

Fig. 4 The graph viewer pushes the operations to the server to be executed in parallel

have submitted many patents on electronics and computing in recent years and that there are many citations between these patent groups.

6 Use case: Patent data analysis

One of our main use cases was the analysis of the US patent data which is available publicly[11]. The data consists of 6 million patents and about 70 million citations between these patents. Patents and citations have several properties such as author, submission year and so on. We have first transformed this relational data into the graph-based format which can be read by GRADOOP. Then, we considered several ways to analyze the data and extract useful information.

Fig. 1b shows an analysis workflow for the patent data. In the example, an analyst wants to find so-called patent families, special structures in the graph that result from the practice of companies not to claim a single patent for a new technology, but to create multiple patents with slight variations. The goal of this behaviour is to keep rivals from easily patenting an almost identical invention. When analyzing the graph, these families are cluttering the view and make it harder to understand relations and patterns. Therefore, it is desirable to find these families and to replace them with placeholders before further analyses are conducted.

In step 1 of the workflow in Fig. 1, the user tries to obtain an overview to the input graph, and attaches a *Big Graph*

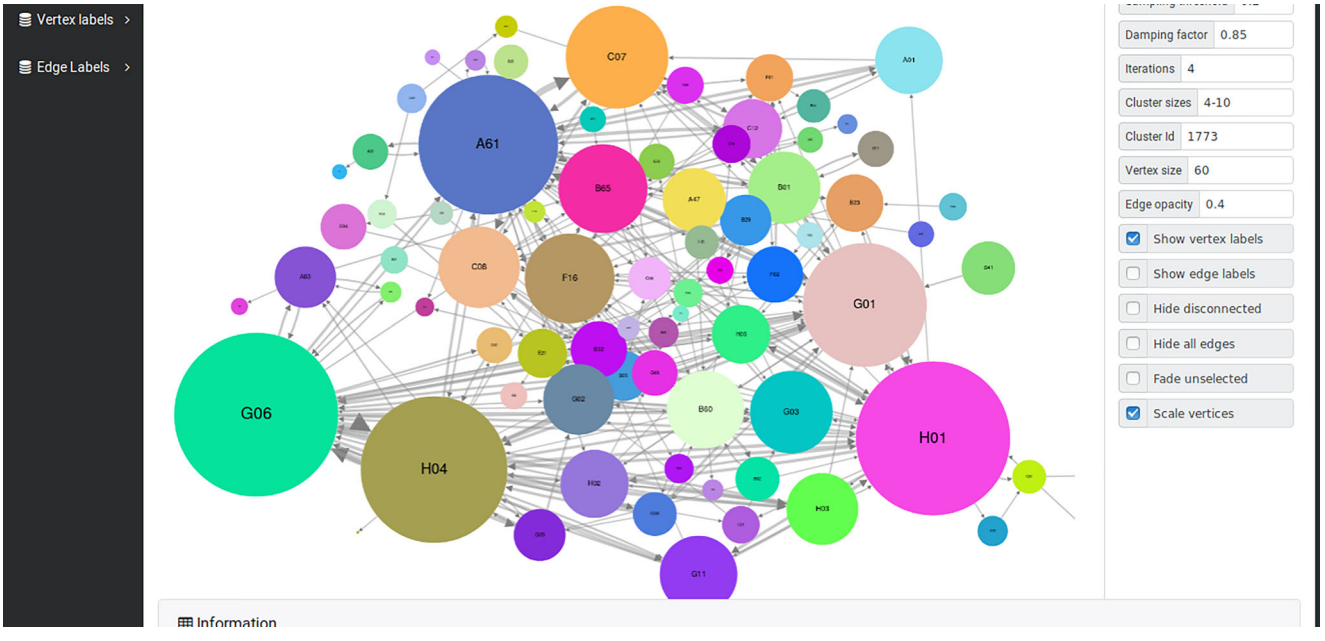

Fig. 5 Graph visualization in the Big Graph View in KNIME-grouping of patent data

[11] http://www.patentsview.org/.

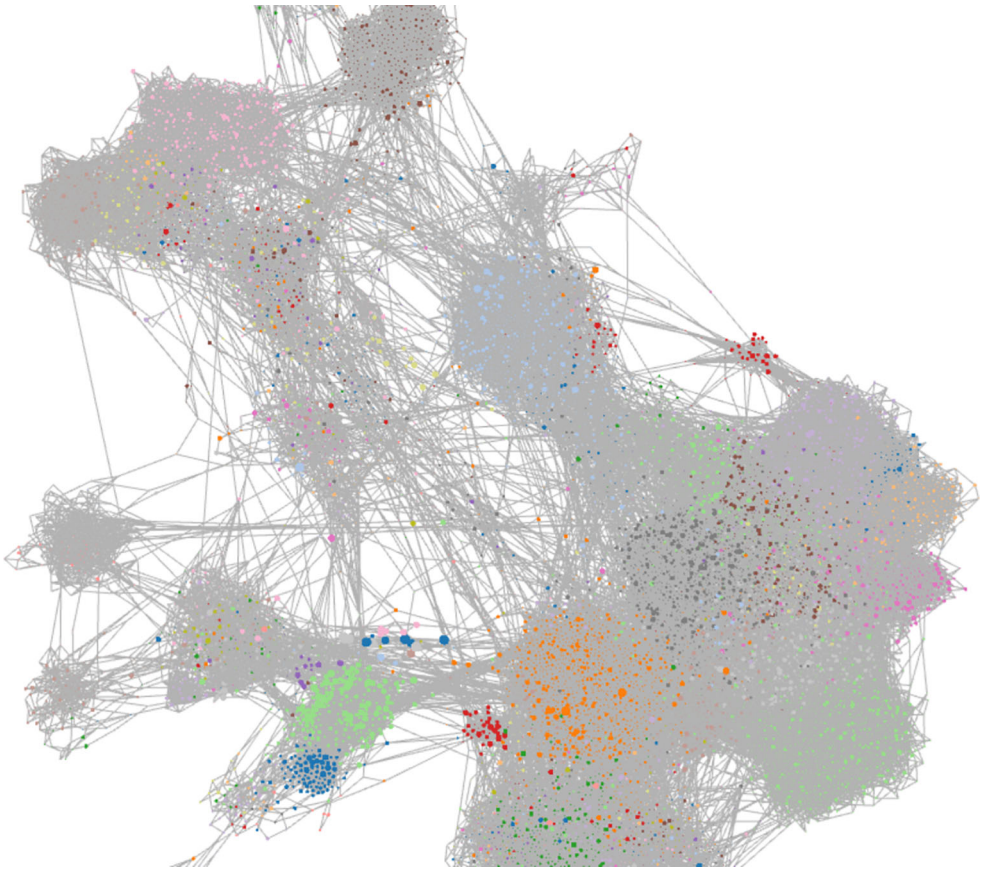

Fig. 6 Sampled graph visualizing patent families

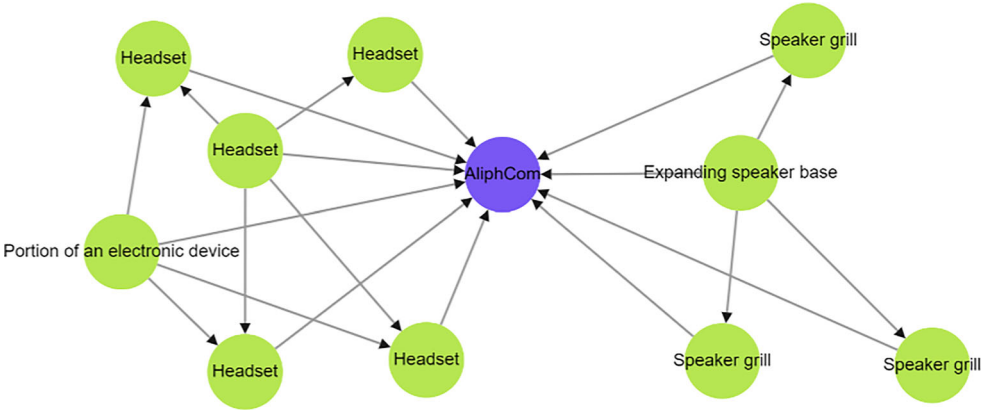

Fig. 7 Exemplary patent family

Viewer to the CSV input node. After executing the initial workflow and opening the graph viewer, the user chooses to layout the graph with a force-embedded layout [30] and applies a vertex sampling which is automatically executed on the server. The result is visualized and by coloring vertices based on specific property values some cliques can be identified which likely are patent families (see Fig. 6).

In the second workflow part, the user further tests her assumptions on the nature of patent families and starts by applying the operator pipeline shown in the lower part of Fig. 1 (right): after the graph is loaded from the distributed file system, a match query is used to identify patents from the same assignee (issuing company) that are applied for in the same year and have a citation relation (derivatives cite the original invention). The following nodes in the workflow serve the purpose of aggregating the related patents (the families) and transform them into several logical graphs (each graph is a family). Because of the large number of patent families, the analyst applies random sampling to make the graph suitable for visualization and man-

ual inspection. After the successful execution of this (sub-) workflow, the results can be checked by looking at them with the graph viewer. This aides the analyst in the process of verifying or changing the initial match pattern, because logical errors can be identified easily. If the workflow is complete, looking at the result may verify the hypotheses or lead to further insights in the nature of the patent graph and a refinement of operations to identify patent families.

After the analyst is satisfied with the results of this task, the upper path in the workflow (step 3) of Fig. 1 is executed: the patent families in the original graph are replaced with a single node representing a whole family and the citations are updated so that they refer to the new nodes. This is done with the "fusion" operator and the non-sampled graph collection of families. Executing this path of the workflow results in a smaller, less complex graph that can be further analyzed. Fig. 7 shows an exemplary patent family that was identified and that occurred in the result of a query on recent highly-cited patents.

In our case, the graph is written to storage. To perform these operations with other tools (e.g. Java), the analyst would have to write the workflows "blindly" (without being able to check intermediate steps), store the results, and validate them in a separate graph visualization tool which would have to be able to interpret the GRADOOP format.

7 Conclusions

We presented the BIGGR approach to achieve a user-friendly specification and execution of large-scale graph analysis workflows. For this purpose, the graph operators of the distributed graph processing system GRADOOP have been integrated into the KNIME data analysis platform. Furthermore, a number of graph visualizations have been added to support a flexible and explorative analysis of large graphs and analysis results. Both, graph operators and visualization operations can be executed on a shared-nothing cluster to support short execution times and scalability to large data volumes. Initial applications showed the functionality and high usefulness of the approach. It is planned to make the described extensions freely available within an upcoming release of KNIME.

In future work, we plan to comprehensively evaluate and optimize the new graph capabilities in KNIME and use them in additional applications. We also work on extensions to GRADOOP for graph transformations and data integration that will also become candidates for inclusion into KNIME.

Acknowledgements The BIGGR project is joint work with KNIME and we thank Tobias Kötter und Mark Ortmann for assistance with technical parts of KNIME.

Funding This work was funded by the German Federal Ministry of Education and Research within the projects BIGGR (BMBF 01IS16030B) and ScaDS Dresden/Leipzig (BMBF 01IS14014B).

References

1. Junghanns M, Petermann A, Neumann M, Rahm E (2017) Management and analysis of big graph data: current systems and open challenges. In: Handbook of big data technologies. Springer, Berlin, Heidelberg, pp 457–505 https://doi.org/10.1007/978-3-319-49340-4-14
2. Junghanns M, Petermann A, Gómez K, Rahm E (2015) Gradoop: scalable graph data management and analytics with Hadoop. arXiv preprint 150600548
3. Junghanns M, Kiessling M, Teichmann N, Gómez K, Petermann A, Rahm E (2018) Declarative and distributed graph analytics with GRADOOP. PVLDB 11:2006–2009. https://doi.org/10.14778/3229863.3236246
4. Rahm E, Nagel WE, Peukert E, Jäkel R, Gärtner F, Stadler PF, Wiegreffe D, Zeckzer D, Lehner W (2019) Big Data competence center ScaDS Dresden/Leipzig: Overview and selected research activities. Datenbank Spektrum 19(1). https://doi.org/10.1007/s13222-018-00303-6
5. Junghanns M, Petermann A, Teichmann N, Gómez K, Rahm E (2016) Analyzing extended property graphs with Apache Flink. In: Proc. ACM SIGMOD Workshop on Network Data Analytics (NDA). https://doi.org/10.1145/2980523.2980527
6. Junghanns M, Kiessling M, Averbuch A, Petermann A, Rahm E (2017) Cypher-based graph pattern matching in GRADOOP. In: Proc. 7th Int. Workshop on Graph Data Management Experiences & Systems (GRADES). https://doi.org/10.1145/3078447.3078450
7. Junghanns M, Petermann A, Rahm E (2017) Distributed grouping of property graphs with GRADOOP. In: Proc. Database systems for Business, Technology and Web (BTW), pp 103–122
8. Petermann A, Junghanns M, Rahm E (2017) DIMSpan: Transactional frequent subgraph mining with distributed in-memory dataflow systems. In: Proc. 4th IEEE/ACM Int. Conf. on Big Data Computing, Applications and Technologies (BDCAT), pp 237–246 https://doi.org/10.1145/3148055.3148064
9. Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, Ohl P, Thiel K, Wiswedel B (2009) KNIME-the Konstanz information miner: version 2.0 and beyond. ACM SIGKDD Explor Newsl 11(1):26–31. https://doi.org/10.1145/1656274.1656280
10. Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y (2006) Scientific workflow management and the Kepler system: Research articles. Concurr Comput Pract Exper 18(10):1039–1065. https://doi.org/10.1002/cpe.994
11. Hofmann M, Klinkenberg R (2013) Rapidminer: data mining use cases and business analytics applications. Chapman & Hall/CRC, Boca Raton, FL
12. Afgan E, Baker D, van den Beek M, Blankenberg D, Bouvier D, Cech M, Chilton J, Clements D, Coraor N, Eberhard C, Grüning BA, Guerler A, Hillman-Jackson J, Kuster GV, Rasche E, Soranzo N, Turaga N, Taylor J, Nekrutenko A, Goecks J (2016) The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. Nucleic Acids Res. https://doi.org/10.1093/nar/gkw343
13. da Silva RF, Filgueira R, Pietri I, Jiang M, Sakellariou R, Deelman E (2017) A characterization of workflow management systems for extreme-scale applications. Future Gener Comput Syst. https://doi.org/10.1016/j.future.2017.02.026
14. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, Soiland-Reyes S, Dunlop I, Nenadic A, Fisher P, Bhagat J, Belhajjame K, Bacall F, Hardisty A, Nieva de la Hidalga A,

Balcazar Vargas M, Sufi S, Goble C (2013) The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Res 41:W557–561. https://doi.org/10.1093/nar/gkt328
15. Grunzke R, Jug F, Schuller B, Jäkel R, Myers G, Nagel WE (2016) Seamless HPC integration of data-intensive KNIME workflows via UNICORE. In: Euro-Par Workshops. Lecture Notes in Computer Science, vol 10104. Springer, Berlin, Heidelberg, pp 480–491 https://doi.org/10.1007/978-3-319-58943-5-39
16. Riazi S, Norris B (2016) Graphflow: Workflow-based big graph processing. In: 2016 IEEE Int. Conf. on Big Data, pp 3336–3343 https://doi.org/10.1109/BigData.2016.7840993
17. Riazi S (2016) SparkGalaxy: Workflow-based Big Data processing. http://www.cs.uoregon.edu/Reports/DRP-201603-Riazi.pdf. Accessed 1 Mar 2019 (directed Research Proposal)
18. Herman I, Melançon G, Marshall MS (2000) Graph visualization and navigation in information visualization: a survey. IEEE Trans Vis Comput Graph 6(1):24–43. https://doi.org/10.1109/2945.841119
19. Bikakis N, Sellis TK (2016) Exploration and visualization in the web of big linked data: a survey of the state of the art. CoRR abs/1601.08059
20. Caldarola EG, Picariello A, Rinaldi A, Sacco M (2016) Exploration and visualization of big graphs – the DBpedia case study. In: Proc. 8th Int. Conf. on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR) https://doi.org/10.5220/0006046802570264
21. Jugel U, Jerzak Z, Hackenbroich G, Markl V (2016) VDDA: automatic visualization-driven data aggregation in relational databases. VLDB J 25(1):53–77. https://doi.org/10.1007/s00778-015-0396-z
22. Rodriguez M, Neubauer P (2010) Constructions from dots and lines. Bull Am Soc Inf Sci Technol 36(6):35–41
23. Rodriguez M, Neubauer P (2012) The graph traversal pattern. In: Graph Data Management: Techniques and Applications IGI Global, pp 29–46
24. Kricke M, Peukert E, Rahm E (2019) Graph data transformations in gradoop. Proc BTW conf.
25. Hudak P (1989) Conception, evolution, and application of functional programming languages. ACM Comput Surv 21(3):359–411. https://doi.org/10.1145/72551.72554
26. Seidman SB (1983) Network structure and minimum degree. Soc Networks 5(3):269–287
27. Giatsidis C, Malliaros FD, Tziortziotis N, Dhanjal C, Kiagias E, Thilikos DM, Vazirgiannis M (2016) A k-core decomposition framework for graph clustering. CoRR abs/1607.02096
28. Hu P, Lau WC (2013) A survey and taxonomy of graph sampling. CoRR abs/1308.5865
29. Rostami MA, Saeedi A, Peukert E, Rahm E (2018) Interactive visualization of large similarity graphs and entity resolution clusters. In: Proc. Extending Database Technology (EDBT) https://doi.org/10.5441/002/edbt.2018.86
30. Kobourov SG (2012) Spring embedders and force directed graph drawing algorithms. Computing Research Repository (CoRR) abs/1201.3011