Community Detection at scale

A comparison study among Apache Spark and Neo4j

Georgios, GK, Kalogeras*
Distributed Intelligent Systems and Data Lab, Department of Electrical and Computer Engineering, University of Peloponnese, Greece
g.kalogeras@go.uop.gr

Vassilios, VT, Tsakanikas
Distributed Intelligent Systems and Data Lab, Department of Electrical and Computer Engineering, University of Peloponnese, Greece
v.tsakanikas@go.uop.gr

Ioannis, IB, Ballas
Distributed Intelligent Systems and Data Lab, Department of Electrical and Computer Engineering, University of Peloponnese, Greece
i.ballas@go.uop.gr

Vassilios, VA, Aggelopoulos
Distributed Intelligent Systems and Data Lab, Department of Electrical and Computer Engineering, University of Peloponnese, Greece
v.aggelopoulos@go.uop.gr

Vassilios, VT, Tampakas
Distributed Intelligent Systems and Data Lab, Department of Electrical and Computer Engineering, University of Peloponnese, Greece
tampakas@uop.gr

ABSTRACT

The proliferation of data generation devices, including IoT and edge computing has led to the big data paradigm, which has considerably placed pressure on well-established relational databases during the last decade. Researchers have proposed several alternative database models in order to model the captured data more efficiently. Among these approaches, graph databases seem the most promising candidate to supplement relational schemes. Within this study, a comparison is performed among Neo4j, one of the leading graph databases, and Apache Spark, a unified engine for distributed large-scale data processing environment, in terms of processing limits. More specifically, the two frameworks are compared on their capacity to execute community detection algorithms.

CCS CONCEPTS

• **Distributed computing methodologies**;

KEYWORDS

graph database, Apache Spark, Neo4j, Label Propagation algorithm

ACM Reference Format:
Georgios, GK, Kalogeras, Vassilios, VT, Tsakanikas, Ioannis, IB, Ballas, Vassilios, VA, Aggelopoulos, and Vassilios, VT, Tampakas. 2022. Community Detection at scale: A comparison study among Apache Spark and Neo4j. In *26th Pan-Hellenic Conference on Informatics (PCI 2022), November 25–27, 2022, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3575879.3575961

PCI 2022, November 25–27, 2022, Athens, Greece

ACM ISBN 978-1-4503-9854-1/22/11…$15.00
https://doi.org/10.1145/3575879.3575961

1 INTRODUCTION

Networks are one of the most prominent ways of representing data structures comprising entities and their relationships. In fact, such is their usefulness that they are deployed across a wide variety of disciplines, including but not limited to technology and the internet, transport, biology, communication and social life [1]. Using graphs to represent these networks offers a good way not only to visually inspect and depict them, but also to apply graph algorithms for acquiring analytics. A graph can help easily visualize an important property of some networks, namely that they are very rarely homogeneous. This means that their internal structure has areas where edges are denser compared to the ones connecting to other parts of the network. This feature represents what is called "community structure" [2].

Exploring a graph and finding hidden patterns within its structure is a very important research topic, known more commonly as community detection, or graph clustering [3]. Finding communities in real life networks has practical implementations. For example, resource management in a large computer network, identification of customers with similar buying behavior and finding social media clusters with a strong internal influence between members are important real life issues for analysts [4].

Since we are living in the era of Big Data, it goes without saying that networks have to depict a huge amount of information, leading to enormous graphs with millions and even billions of vertices in some cases [4]. This fact alone makes community detection a very difficult endeavor, that is very demanding in terms of both time and computational resources. The need for novel approaches that can perform better than typical sequential and centralized community detection algorithms is now more evident than ever.

The scope of this paper is to present a comparative analysis of Neo4j and Apache Spark for community detection algorithms on large graphs. The rest of this paper is structured as follows. Section 2 presents the state of the art on community detection. Section 3 presents the background of the problem and commonly used algorithms. Section 4 gives details on the experiments carried out.

Sections 5 and 6 present the experiment results and conclusion respectively.

2 STATE OF THE ART

Community detection can be implemented in a variety of real-world applications where data can be depicted in the form of a network graph. Biology is one scientific field where community detection is used to identify metabolic processes and protein interactions [4]. Cluster analysis in complex brain networks is used in neuroscience for scientists to uncover specific areas that work as sub-divisions of the human brain [1]. Other areas such as financial sciences, transport and even computer vision and image processing also benefit from the process of community detection. However, perhaps the most prominent field where community detection is used is in social networks. This of course also affects social media, which have been evolved with an exponential growth in recent years. Communities in social media are important for highlighting relationships between individuals or entities interacting throughout the web.

Initial community detection algorithms were designed to run sequentially and performed adequately well for small networks. Traditional methods involve algorithms based on graph partitioning or take a hierarchical approach. Some of the most widely used algorithms follow divisive or modularity based approaches to tackle the task of community detection [5]. Two approaches have been followed to create more efficient algorithms that can handle the complexity of large scale graphs.

Parallel approach aims to utilize as much of the CPU time as possible by breaking down calculations into individual parts that can be executed at the same time e.g., in parallel. Different processors or cores can work independently and proceed with calculations on data over a shared memory, cutting down on the time required for community detection. Commonly used algorithms like the modularity based Girvan and Newman [6] have been optimized to run in parallel systems to offer better performance. Graph partitioning is a common approach that splits the graph into smaller parallelizable parts to reduce complexity [3]. Parallel frameworks like OpenMP or hybrid CPU-GPU-CUDA approaches are often used to implement sequential algorithms for better efficiency [7], [8]. Algorithms based on the distance dynamic concept have also attracted research interest for implementing them in a parallel manner with increased convergence, through a sub-network partitioning procedure [9].

The second approach involves distributed architectures, where computation is performed over several different computers. Here, the strengths of distributed frameworks such as Hadoop and Spark are utilized to improve computational efficiency and address scaling issues. Again well-known algorithms like the Girvan-Newman are implemented in the aforementioned frameworks and take full advantage of MapReduce functions for better community detection [10]. Gawande *et al.* developed cuVite a distributed-memory multi-GPU implementation of Louvain method on heterogeneous systems which shows the relative performance of cuVite with the state-of-the-art shared and distributed-memory CPU-only as well as GPU-based implementations [12].

In 2008, Blondel *et al.* introduced a multi-phase, multi-iteration heuristic for modularity maximization called the Louvain method [11]. Owing to its speed and ability to yield high quality communities, the Louvain method continues to be one of the most widely used tools for serial community detection [12].

Zeng and Yu developed a scalable distributed community detection algorithm for large graphs based on the Louvain method which shows a superior scalability and accuracy over the existing distributed Louvain algorithms. Using the vertex delegates partitioning, the algorithm can achieve balanced workload and communication among massive processors with large graphs. Moreover, they develop a collective operation to synchronize the information on delegated vertices, and designed a new heuristic strategy to ensure the convergence of the distributed clustering algorithm [13].

A variety of static community detection algorithms and quality metrics were developed in the past few years. Recent works in community detection focus in distributed community detection algorithms in dynamic graphs. Abughofa et al. propose a node-grained Incremental Distributed Weighted Community Clustering (IDWCC) for maintaining a dynamic graph over streaming data. The experiments showed that IDWCC outperforms DWCC for large dynamic graphs. IDWCC produced the same or better WCC values compared to DWCC. It was also two to three times faster than DWCC [14].

Another algorithm widely used for community detection is Label Propagation Algorithm (LPA) [15], due to its fast performance and low complexity [16]. The research community has put effort into improving the algorithm for better handling of big data networks. Parallel approaches have shown significant improvement in time consumption while maintaining the detection quality [17], [18]. Distributed frameworks have also been used with LPA implementations to improve the scalability of the algorithm [19].

It is clear that through utilization of powerful parallel / distributed frameworks , very good results can be obtained in terms of computation time and complexity. Parallel tools like OpenMP and graph databases like Neo4J offer researchers important optimization opportunities, while distributed frameworks like Spark provide easy scaling of big data applications. Hybrid approaches can also be utilized to further improve performance of sequential algorithms.

3 BACKGROUND

3.1 Apache SPARK Project

Spark was launched by Matei Zaharia at UC Berkeley's AMPLab in 2009 and open sourced in 2010. Thenceforth, various research initiatives have contributed for enhancing and improving Spark core and its main kernel libraries [20]. Among the most successful ones, Spark's MLlib developed by Mlbase project and supported by KeystoneML. Spark SQL begun from Shark project before becoming a primary library in Apache Spark. Also, GraphX started as a research project at the AMPLab. Later, it became a part of the Apache Spark project. Many packages have also been contributed to Apache Spark from both academia and industry. Apache Spark system consists of several main components including Spark core and upper-level libraries: Spark's MLlib for machine learning, GraphX for graph analysis, Spark Streaming for stream processing and Spark SQL for structured data processing. It is evolving rapidly with changes

Table 1: Datasets used for the experiments phase

Name	Type	Nodes	Edges	Communities	Description
com-DBLP	Undirected, Communities	317,080	1,049,866	13,477	DBLP collaboration network
com-Youtube	Undirected, Communities	1,134,890	2,987,624	8,385	Youtube online social network
com-LiveJournal	Undirected, Communities	3,997,962	34,681,189	287,512	LiveJournal online social network

to its core APIs and addition of upper-level libraries. Its core data abstraction, the Resilient Distributed Dataset (RDD), opens the door for designing scalable data algorithms and pipelines with better performance. With the RDD's efficient data sharing and a variety of operators, different workloads can be designed and implemented efficiently. While RDD was the main abstraction introduced in Spark 1.0 through the RDD API, the representation of datasets has been an active area of improvement. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood [21]. GraphFrames provide DataFramebased Graphs. They aim to provide both the functionality of GraphX and extended functionality taking advantage of Spark DataFrames. GraphFrames generalize the ideas behind GraphX [22] and Vertexica [23] by maintaining arbitrary views of a graph (e.g., triplets or triangles) and executing queries using joins across them.

3.2 NEO4J Project

Neo4j is considered the leading software in the area of Graph Databases. Neo4j is an open-source graph database implemented in Java, and it was released back in 2007. Neo4j is a fully transactional database based on a persistent Java engine with the capacity to store structures in the form of graphs. The main features of Neo4j are [24]: (i) Horizontal scalability (in the Enterprise version) and vertical scalability (in the Community version). (ii) A new efficient query language (Cypher), (iii) Storage that is disk - based – through proprietary file systems and (iv) Guaranteed integrity according to ACID properties. Neo4j can be deployed under two modes, Desktop mode and Cluster mode. Desktop refers to a standalone installation, where all the data are stored in a single machine. On the other hand, Cluster mode offers a distributed environment with the capacity of fault tolerance, transparent scaling and causal consistency. A lot of research works [25] suggest that graph databases are the closest candidates to substitute SQL based systems in specific areas, like healthcare [26] and social media analysis [27], [28], [29], [30].

3.3 Community Detection Algorithms

In order to compare community detection while scaling up to accommodate big data applications, we have opted for using the Label Propagation Algorithm (LPA) [15]. LPA draws inspiration from epidemic spreading and utilizes node labels to detect communities [16]. As labels propagate through the network, communities are formed when a large number of nodes reach consensus on a particular label. The choice of the LPA was made for two reasons: a low computational complexity and availability for both frameworks used in the experiments (Spark and Neo4j).

4 DESIGN OF EXPERIMENTS

4.1 Explored Datasets

A set of datasets have been selected in order to perform the comparison study. All datasets were acquired by the "SNAP – Stanford Large Network Dataset Collection" [31]. Table 1 provides all the important information related to the selected datasets.

4.2 Experiment's Infrastructure

This paragraph presents in detail the infrastructure used for the execution of the experiments.

4.2.1 Physical Environment. The physical environment of the infrastructure used to deploy a VM-hosted Hadoop cluster comprised of five (5) computer nodes (DELL©), with the following technical specifications: CPU: 6 cores (6 threads) / i5-8500 CPU @ 3.00GHz, RAM: 32GB (DDR4 2666MHz), HD type: SATA 6Gb/s HDD, and one computer node, with the following technical specifications: CPU: 8 cores (16 threads)/ i7-11700F @ 2.50GHz, RAM: 64GB (DDR4 3200MHz), HD type: SATA 6Gb/s SDD. The nodes were connected under a star LAN topology (switch speed: 1Gbps).

4.2.2 Apache Spark configuration. The experiments were performed on a VM-Hosted Hadoop cluster consisting of a master node, and five worker (slave) nodes with the following resources: Slave nodes deployed as VMs (5 cores/node, RAM 27.7GB/node and max container RAM size 26.2GB), Master node (deployed as VM, with 14 vcores and 58 GB RAM). Spark applications were submitted from a client node that was connected to the same local area network.

We leverage the HDFS of Apache Hadoop to save and retrieve the input data and the results. Each application instance in YARN has its own Application Master (AM) process that runs in the first YARN resource container that Resource Manager started for that application on a slave node. We have chosen the cluster deploy mode for our experiments. In cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application [32].

The software configuration per node comprises of Apache HDFS/YARN (version Hadoop3.2.1), Apache Spark (version 3.1.2),

Table 2: Spark explored configurations

Environment	Cores/Executor	Memory/Executor (MB)	Executors	Cores for all Executors	Total Executors Memory (MB)	Cores/Application Master (driver)	Driver memory (MB)
conf1	1	5760	20	20	115200	1	1024
conf2	2	11520	10	20	115200	1	1024
conf3	4	23040	5	20	115200	1	1024

Virtualization Platform VMWare Workstation 16 Player, Operating System Ubuntu (version 20.04 LTS).

Using the same number of nodes in the cluster we experimented in different scenarios by changing the number of cores per executor while allocating the same total resources. In Table 2, the different Spark configurations considered in the study are detailed.

4.2.3 Neo4j configuration. The experiments were performed on the VM that previously was described as the Master node of the VM-hosted Hadoop cluster. Neo4j Desktop Version 1.4.15 is used with Graph Data Science Library 2.2.0 and APOC 4.4.0.0 plugins respectively installed. For the first graph database which stores the com-DBLP dataset (dblp ungraph), the following settings have been applied:

- heap.initial_size=4G, heap.max_size=4G.
- pagecache.size=512m, all under the dbms.memory.pagecache.

For the second graph database which stores the com-Youtube dataset (youtube ungraph), the following settings have been applied :

- heap.initial_size=8G, heap.max_size=8G.
- pagecache.size=512m, all under the dbms.memory.pagecache.

For the third and last graph database which stores the com-LiveJournal dataset (lj ungraph), the following settings have been applied :

- heap.initial_size=44G, heap.max_size=44G.
- pagecache.size=512m, all under the dbms.memory.pagecache

Finally, the apoc.cypher.runMany library has been utilized, in order to execute the LP algorithm in parallel.

5 RESULTS

This section presents the results of the experimental phase of the study. More specifically, Figure 1 presents the total execution time for the LP algorithm to be executed in all setups of the Apache Spark framework, aiming to identify the most appropriate configuration for calculating communities on the different datasets. Experiments for each configuration and for each dataset were conducted under a 10-fold rationale. Average execution time, along with the standard deviations are presented.

Based on the results presented in Figure 1, the most appropriate configuration for the Apache Spark framework was selected, for each one of the considered datasets. More specifically, conf3

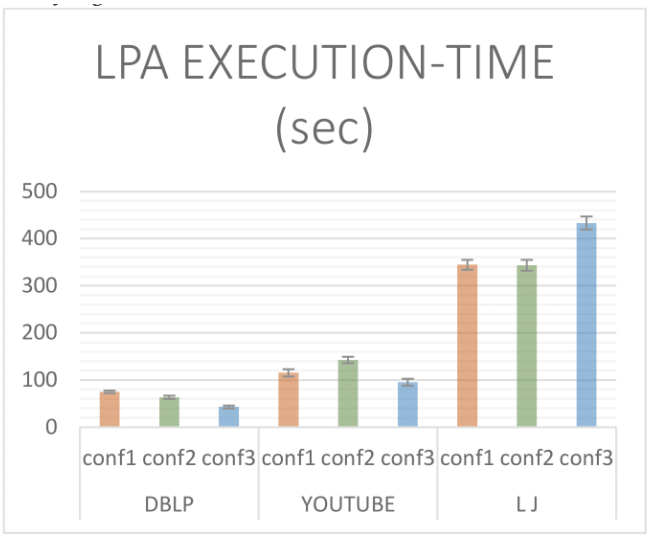

Figure 1: Required execution time for the LP algorithm, on the different Spark configurations.

presented the optimum behavior for both DBLP and YOUTUBE datasets, while conf2 for the LiveJournal dataset.

Figure 2 presents the relative results on the average execution time, as observed on the two considered systems. In detail, one can observe that Neo4j framework outperforms Apache Spark for DBLP and YOUTUBE datasets, while Apache Spark presents significantly improved results on the LJ dataset.

As far as the consumed memory is concerned, Figure 3 presents the results of the occupied RAM memory for both Apache Spark and Neo4j. It is important to mention that the reported values on Spark refer to the total consumed RAM on all involved nodes. As one can observe, Spark's memory footprint is considerably greater than the relative of Neo4j, regardless of dataset size.

Regarding the evaluation metrics on Neo4j, it is mentioned that the execution time of the LP algorithm includes the loading of the data from the SNAP file, the creation of the graph structure as well as the saving of the results to the hard drive. We utilized the neo4j-admin tool, which is proposed by Neo4j for loading large datasets.

When combing the results of Figures 2 and 3, one can conclude that when the available RAM memory of Neo4j is adequate to load the processed dataset, then we can achieve improved execution

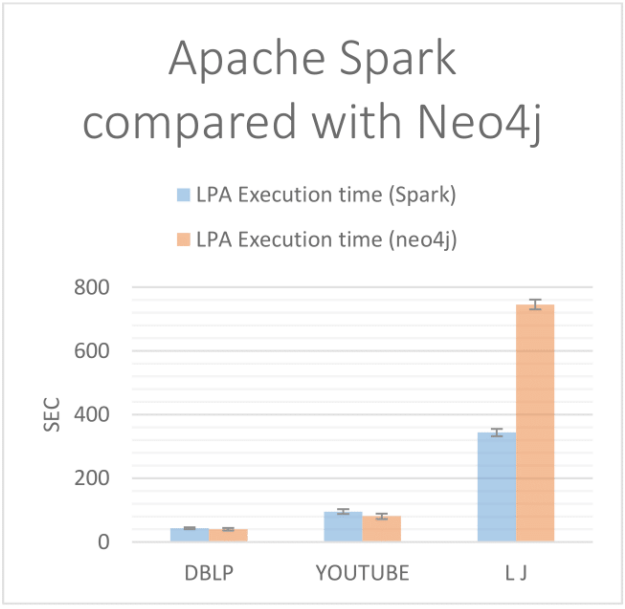

Figure 2: Comparison of the required execution time for the LP algorithm on both Apache Spark and Neo4j frameworks.

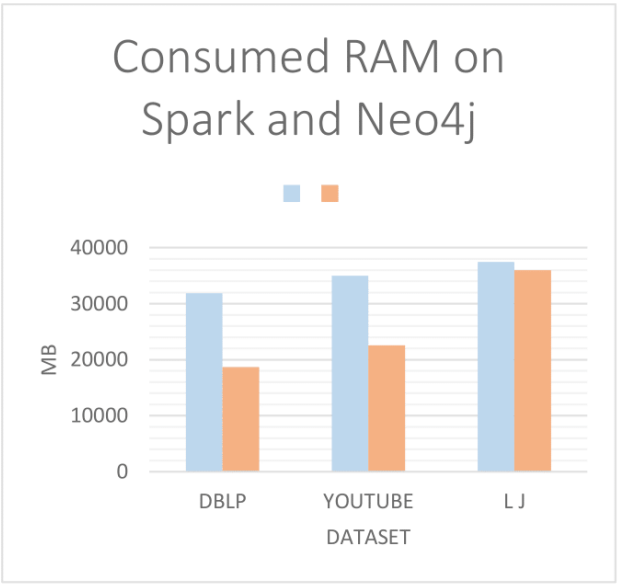

Figure 3: Comparison of the consumed memory between Spark and Neo4j frameworks.

times with lower consumed RAM, when compared to a distributed framework like Apache Spark. Yet, when the processed dataset is greater than the available RAM, Spark outperforms Neo4j in terms of execution time.

6 CONCLUSIONS AND DISCUSSION

Graph databases constitute a novel technology for storing data. This offers the possibility of executing graph related algorithms, like shortest path algorithm, traversals and link analysis directly on the stored data, thus avoiding the transformation of the data to other formats. This study attempts to compare a unified engine for distributed large-scale data processing (Spark) with a graph database (Neo4j) for executing a community detection algorithm (Label Propagation).

The experiments revealed in general, that Neo4j engine outperformed the Apache Spark cluster for most used datasets. Yet, when the size of the processing dataset exceeds a certain threshold, Spark outperformed the processing engine of Neo4j.

On the same page, our experience on setting up the two environments showed that Neo4j has an easier –to – use interface. The installation process was by far easier and more straightforward, when comparing it with Apache Spark installation.

In conclusion, Neo4j is a great environment for storing data, which is though limited to the physical memory of the deployment server. When more data need to be processed, a distributing environment, like Apache Spark, is required in order to scale up the final solution.

As future steps related to this work, we aim to test other type of algorithms, like traversals, and evaluate the performance of discussed environments. Finally, we intend to evaluate solutions (e.g., Neo4j Connector for Apache Spark) which integrate Neo4j and Apache to a single solution, in order to be able to harvest the benefits of both systems.

ACKNOWLEDGMENTS

The work presented in this paper is supported by the inPOINT project (https://inpoint-project.eu/), which is co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (Project id:T2EDK_04389).

REFERENCES

[1] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics reports*, vol. 533, no. 4, pp. 95–142, 2013.
[2] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.
[3] J. Zeng and H. Yu, "A study of graph partitioning schemes for parallel graph community detection," *Parallel Computing*, vol. 58, pp. 131–139, 2016.
[4] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3–5, pp. 75–174, 2010.
[5] D. Naik, D. Ramesh, A. H. Gandomi, and N. B. Gorojanam, "Parallel and distributed paradigms for community detection in social networks: A methodological review," *Expert Systems with Applications*, vol. 187, p. 115956, 2022.
[6] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.
[7] M. Alandoli, M. Shehab, M. Al-Ayyoub, Y. Jararweh, and M. Al-Smadi, "Using gpus to speed-up fcm-based community detection in social networks," in *2016 7th international conference on computer science and information technology (CSIT)*, 2016, pp. 1–6.
[8] M. Alandoli, M. Al-Ayyoub, M. Al-Smadi, Y. Jararweh, and E. Benkhelifa, "Using dynamic parallelism to speed up clustering-based community detection in social networks," in *2016 IEEE 4th international conference on future internet of things and cloud workshops (FiCloudW)*, 2016, pp. 240–245.
[9] T. He, L. Cai, T. Meng, L. Chen, Z. Deng, and Z. Cao, "Parallel community detection based on distance dynamics for large-scale network," *IEEE Access*, vol. 6, pp. 42775–42789, 2018.

[10] S. Moon, J.-G. Lee, and M. Kang, "Scalable community detection from networks by computing edge betweenness on mapreduce," in *2014 International Conference on Big Data and Smart Computing (BIGCOMP)*, 2014, pp. 145–148.
[11] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, É. Lefebvre, and others, "The Louvain method for community detection in large networks," *J of Statistical Mechanics: Theory and Experiment*, vol. 10, p. P10008, 2011.
[12] N. Gawande, S. Ghosh, M. Halappanavar, A. Tumeo, and A. Kalyanaraman, "Towards scaling community detection on distributed-memory heterogeneous systems," *Parallel Computing*, vol. 111, p. 102898, 2022.
[13] J. Zeng and H. Yu, "A scalable distributed louvain algorithm for large-scale graph community detection," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 268–278.
[14] T. Abughofa, A. A. Harby, H. Isah, and F. Zulkernine, "Incremental Community Detection in Distributed Dynamic Graph," in *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*, 2021, pp. 50–59.
[15] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.
[16] S. E. Garza and S. E. Schaeffer, "Community detection with the label propagation algorithm: a survey," *Physica A: Statistical Mechanics and its Applications*, vol. 534, p. 122058, 2019.
[17] J. Soman and A. Narang, "Fast community detection algorithm with gpus and multicore architectures," in *2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, pp. 568–579.
[18] E. Jokar and M. Mosleh, "Community detection in social networks based on improved label propagation algorithm and balanced link density," *Physics Letters A*, vol. 383, no. 8, pp. 718–727, 2019.
[19] Q. Zhang, Q. Qiu, W. Guo, K. Guo, and N. Xiong, "A social community detection algorithm based on parallel grey label propagation," *Computer Networks*, vol. 107, pp. 133–143, 2016.
[20] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007, pp. 59–72.
[21] "Spark SQL and DataFrames - Spark 3.3.0 Documentation." https://spark.apache.org/docs/latest/sql-programming-guide.html (accessed Oct. 12, 2022).
[22] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "${$GraphX$}$: Graph Processing in a Distributed Dataflow Framework," in *11th USENIX symposium on operating systems design and implementation (OSDI 14)*, 2014, pp. 599–613.
[23] A. Jindal, P. Rawlani, E. Wu, S. Madden, A. Deshpande, and M. Stonebraker, "Vertexica: your relational friend for graph analytics!," 2014.
[24] R. V. Bruggen, *Learning Neo4j*. Packt Publishing, 2015.
[25] M. Kendea, V. Gkantouna, A. Rapti, S. Sioutas, G. Tzimas, and D. Tsolis, "Graph dbs vs. column-oriented stores: A pure performance comparison," in *International Workshop on Algorithmic Aspects of Cloud Computing*, 2015, pp. 62–74.
[26] J. A. Stothers and A. Nguyen, "Can Neo4j replace PostgreSQL in healthcare?," *AMIA Summits on Translational Science Proceedings*, vol. 2020, p. 646, 2020.
[27] A. Virk and R. Rani, "Efficient Approach for Social Recommendations Using Graphs on Neo4j," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018, pp. 133–138.
[28] N. Giarelis, N. Kanakaris, and N. Karacapilidis, "An innovative graph-based approach to advance feature selection from multiple textual documents," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*, 2020, pp. 96–106.
[29] N. Giarelis, N. Kanakaris, and N. Karacapilidis, "On a novel representation of multiple textual documents in a single graph," in *International Conference on Intelligent Decision Technologies*, 2020, pp. 105–115.
[30] N. Giarelis, N. Kanakaris, and N. Karacapilidis, "On the utilization of structural and textual information of a scientific knowledge graph to discover future research collaborations: a link prediction perspective," in *International Conference on Discovery Science*, 2020, pp. 437–450.
[31] "Stanford Large Network Dataset Collection." https://snap.stanford.edu/data/ (accessed Oct. 12, 2022).
[32] "Running Spark on YARN - Spark 3.1.2 Documentation." https://spark.apache.org/docs/3.1.2/running-on-yarn.html (accessed Oct. 12, 2022).