An Analysis on Graph-Processing Frameworks: Neo4j and Spark GraphX

Alabbas Alhaj Ali, Doina Logofătu

An Analysis on Graph-Processing Frameworks: Neo4j and Spark GraphX*

Alabbas Alhaj Ali, Doina Logofatu

Frankfurt University of Applied Sciences, Nibelungenpl. 1, 60318 Frankfurt am Main, Germany logofatu@fb2.fra-uas.de
https://www.informatik.fb2.fh-frankfurt.de/~logofatu

Abstract. Numerous graph algorithms have been developed to address a variety of problems in the industry, ranging from fraud detection to scheduling or even recommendation systems. Graph-processing frameworks are hence created to simplify the implementation of graph-based solutions. Nonetheless, the number of such frameworks has grown significantly over the past decades with varying benefits and drawbacks. Understanding the requirements and characteristics of each framework plays a vital role in the selection of a suitable solution to a given problem. In this work, we evaluate the performance and usability of 2 popular graph-processing frameworks Neo4j and Apache Spark GraphX by implementing a PageRank solution to solve a practical business problem derived from the Yelp dataset.

Keywords: Graph-processing frameworks · Neo4j · Apache Spark GraphX PageRank · Yelp dataset.

1 Introduction

Graph algorithms have always proven to be an efficient problem-solving technique thanks to their natural ability to model real-world domains. Their applications are diverse and numerous, ranging from basic routing problems in transportation/logistic industry [12] to fraud detection in the financial world [33], from recommendation systems in user-oriented services like e-commerce [39] to social media [3], and much more. Efficient as they are, developing a graph-based solution to handle an industrial problem is no trivial task in that graph theory is among the oldest and most complex areas of mathematics and computer science [34].

Furthermore, in 2018, more than 3 billion people used the Internet, 2.5 quintillion bytes of data were produced on a daily basis, and over 90 percent of data in the whole world was generated in a period of two years [1]. The growth was even more impressive especially for online social networking services where graph algorithms always played a dominant role. For example, figure 1 illustrates the dramatic growth of Facebook - a famous social network platform - between 2008

* Supported by the Frankfurt University of Applied Sciences

and 2019. As the data magnitude grows exponentially, developers face new challenges in using graph algorithms. It is no longer sufficient to only be capable of applying them, but we must be able to implement these algorithms efficiently. The work in [20], for instance, introduces numerous scalable variants of graph algorithms such as Minimum Spanning Tree (MST), Breadth-First Search (BFS) and Single-Source Shortest Path (SSSP) algorithms to address several challenges in large-scale classical graph problems.

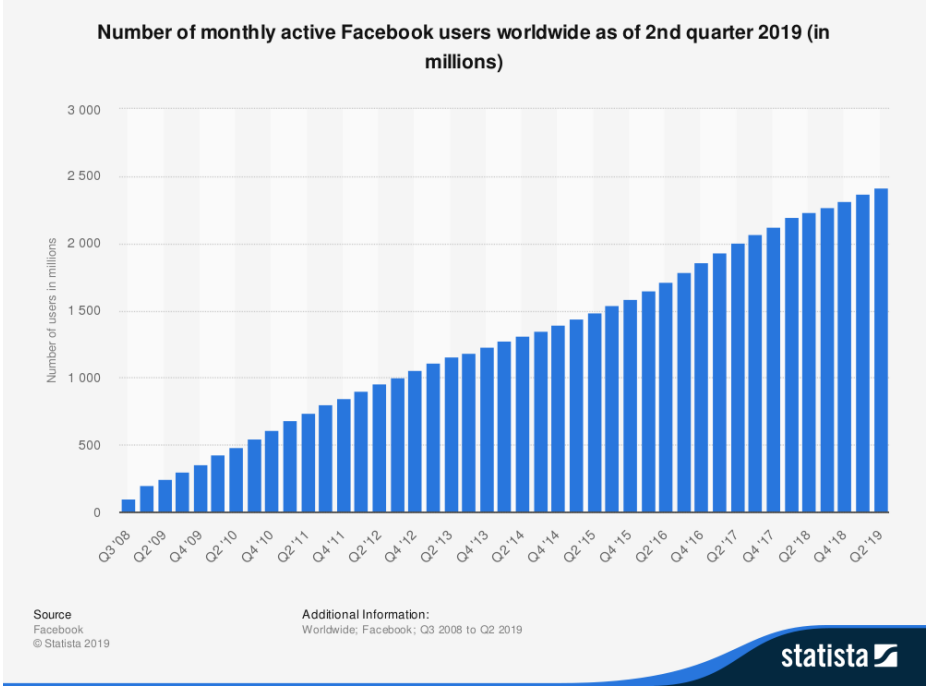

Fig. 1. Facebook users worldwide 2008-2019 [8] Please note that short captions are centered, while long ones are justified by the macro package automatically.

To bridge the gap between academic research and software development, many graph-processing frameworks are created to accelerate the design, implementation and testing of graph-based solutions and their integration into existing systems. These frameworks have grown significantly over the past decades. Each framework of this type provides functionalities either as a graph database (Neo4j [17] and OrientDB [37]) or a graph processing engine (Apache Spark GraphX [13], Pregel [25], BGL [35] and GraphBase [21]) or even both (Giraph [2]). A more thorough review of key players in this domain can be found in [29].

In this study, we conduct an experiment using 2 of the most popular graph-processing frameworks - Neo4j and Spark GraphX - to analyze the Yelp dataset

[43]. Our paper aims at providing future researchers and developers with a good reference about the usability and performance between these 2 libraries. A clear understanding of these characteristics will help them choose the best solution for a given problem.

2 Study Case

2.1 Motivation

One major challenge for all types of businesses is user acquisition. In simpler terms, companies must find a way to attract customers. There are various strategies to address this challenge, from giving coupons to placing advertisements. In this study, we focus on a more traditional method which is to invite influential people to test our services with the intention of attracting their followers. For instance, if Michael Jackson was to use the services provided by our company and put on a good word, his billions of fans would more likely follow his choice and be willing to buy the same services he used. However, those celebrities may be too expensive for small and medium businesses to afford, which leads to the question of how we can find affordable and influential customers from the general population? A similar problem can be found in [19]

With the invention of social media platforms such as Yelp [23], Facebook [36], Twitter [27] or YouTube [16], more and more ordinary people can achieve great fame overnight by creating sharing their inspiring stories. They are normal non-celebrity people who are much more affordable for businesses to invite and still have a huge population of fans. If we have access to some customer data, it is possible to search for these people. In this work, we employ the Yelp dataset for the sake of experiment.

2.2 Yelp Dataset

Yelp is an American company born with a mission to connect people with great local businesses. They create a digital platform that makes it easy for users to provide their reviews and recommendations about local restaurants, shopping or any services from their real-life experience. Yelp has published a subset of information accumulated by their platform since 2013 via their annual machine learning challenge, namely Yelp Challenge [43]. As of Round 10 of the challenge, the dataset includes 9,489,337 users and 156,639 businesses from 10 states in the North America region. There are a total of 1,005,693 tips from users about businesses, 4,736,897 reviews of businesses by users and 35,444,850 friend relationships.

2.3 Problem Description

In this research, we are only interested in users and their friends. From the original dataset, let's construct a friendship network as a graph $G = (V, E)$ with

vertex set V and edge set E. Each vertex represents a user from the Yelp dataset, while each edge represents a bidirectional relationship between 2 users: friendship. Over half a million users without friends are ignored since their influence is assumed to be zero. Table 1 illustrates the scale of this friendship network where WCC stands for Weakly Connected Component [6]. The remaining task is to implement algorithms that input this graph and output the influential users we are searching for.

Table 1. Statistics of the friendship network

Nodes	**Edges**	**WCCs**	**Largest WCC**	
			Nodes	**Edges**
8,981,389	35,444,850	18,512	8,938,630	35,420,520

2.4 Approach

To search for influential people, it is of paramount importance to define a deterministic approach to measure *influence*. Measuring influence of a person by his number of friends is the most natural but naive approach. Practice has demonstrated that such a method will not provide the best candidates. For instance, imagine a network where the top 5 users have at least a million friends each. However, none of these 5 users is a friend to each other, but all of them are friends to a common user X who, strangely, has exactly 5 friends. Is it more beneficial for a business to invite all 5 users or should they invite only X instead? That is the motivation behind algorithms like PageRank [30, 11] whose superiority has been proven in practice.
In our study, influential people are identified by their PageRank score. The same program will be implemented as a query in both Neo4j and Spark GraphX in that these frameworks have out-of-the-box support for PageRank. The implementations and results will be then used to assess the performance and usability of these 2 frameworks.

3 Related works

In the work [10] the author's analyzed a seven graph processing frameworks under five design aspects including distribution policy, on-disk data organization, programming model, synchronization policy and message model. the work reveal some interesting finding that the vertex-cut method over weighs the edge-cut method on neighbor-based algorithms while leads to inefficiency for non-neighbor-based algorithms using asynchronous update can reduce the total workload by 20% to 30% however the processing time may still doubled due to fine-grained lock conflicts.
The work [31] analyse the use of Pregel-like Large Scale Graph Processing

Frameworks in processing the data of Social Network. the work discussed various different methods, compared their run-times, and proved that the graph processing framework is suitable for analysis of social networks. The work concludes by providing several key observations about algorithmic design and framework flexibility.
The work [7] focus in the comparison terms of execution time between second generation frameworks and the de-facto standard Hadoop in processing large-scale graph analysis. the authors selected the k-core algorithm and its adaptation each platform. They found that from a programming paradigm point of view, a vertex centric framework is the better fit for graph related problems. they recommend one of the Pregel-inspired frameworks for graph processing as the performance of vertex-centric frameworks is not matched by Map/Reduce-based frameworks, even by improved ones, such as Stratosphere.
An intensive Analysis of distributed programming models and frameworks for large-scale graph processing has been introduced by the work of *Corbellini, A., Godoy, D., Mateos, C., Schiaffino, S. and Zunino, A.* [4], The work provide an perspective analyse for adopted programming model of popular frameworks designed to process large-scale graphs with the goal of assisting software developers/designers in choosing the most adequate tool.

4 Solution in Neo4j

4.1 Overview

Neo4j is a native graph database that is optimized for storing and processing data relationships. Distinguished from traditional databases where data is usually arranged in rows, columns and tables, Neo4j introduces a flexible structure, namely data record (or node) which stores direct pointers to all the nodes it's connected to. Because Neo4j is designed around this simple, and yet powerful optimization, it enables queries with complexity never before imagined and performance never before thought possible. Most importantly, Neo4j provides all functionalities as normal full-fledged databases and is fully compliant ACID transaction principles [28], making it suitable to use graphs for data in production scenarios.

4.2 Solution

Listing 1 presents a reference implementation for finding N influential users using PageRank in Neo4j. The language of implementation is Cypher [9] which is introduced by Neo4j to describe complex queries. As can be seen, the implementation is quite complicated, and what is more, requirement of learning Cypher may prove to be a challenge for many developers since it is not so popular as a programming language. Albeit Neoj4 offers drivers for multiple popular programming languages such as Java [14], Python [38], .NET [15] and JavaScript [40], Cypher still remains the default language of choice for the majority of Neoj4 developers.

```
CALL algo.pageRank.stream(
    'MATCH (u:User)
    WHERE exists( (u)-[:FRIEND]-() )
    RETURN id(u) as id',

    'MATCH (u1:User)-[:FRIEND]-(u2:User)
    RETURN id(u1) as source, id(u2) as target',

    {graph:'cypher'}
)

YIELD nodeId, score
    with algo.asNode(nodeId) as node, score
    order by score desc limit N

RETURN node.id, score
```

Listing 1: Cypher code for finding top N influential users using PageRank in Neo4j

5 Solution in Spark GraphX

5.1 Overview

A well-known approach to cope with large problem size is to follow the MapReduce paradigm [5] in combination with distributed systems. In other words, the graph data input could be replicated or partitioned across a number of computing nodes. The computation results accumulated across all nodes will then be reduced to one final result. However, the development of a distributed solution is itself no trivial task.

To simplify the development process, a variety of frameworks for distributed programming have been widely used in academic research as well as in industry such as Hadoop or Storm by Apache [26], BigQuery by Google [22], Disco by Nokia [18] and many more. Many, but not all of them, offer a builtin MapReduce programming model. In our experiment, we opt for a higher level programming model that can operates on top of Hadoop environment, namely Apache Spark [45]. It also comes with a dedicated module for graph processing called GraphX [42].

The core of Apache Spark lies in the resilient distributed dataset (RDD) model, a fault-tolerant collection of elements partitioned across the nodes of the cluster that can be operated in parallel [44]. RDD makes extensive use of distributed memory, enabling efficient data reuse and fault tolerance. Its key idea is to log coarse-grained transformation (map, filter, join) to build the dataset rather than the actual data.

GraphX is the graph processing framework that extends Spark data-parallel system with the a graph-parallel computation model called Pregel, without caus-

ing complex joins or excessive movements [41]. Pregel model is a message passing interface which features parallel vertex-centric computation, consisting of a sequence of iterations (or super steps) [25]. During each super step, a user-defined function that defines the behavior at vertex V and superstep S, is invoked. Pregel tackles various challenges in efficient processing of large graphs: they often exhibit irregular relationships, poor locality in data access patterns, unbalanced partitioned tasks and a varying degree of parallelism over the course of execution [24].

5.2 Solution

Listing 2 illustrates a reference implementation for finding influential users using PageRank in Spark GraphX. The language of implementation is Python which is supported by GraphX. Compared to Listing 1, this implementation is much shorter in length and simpler to read. Furthermore, Python is one of the most popular programming language widely used all around the world and well-known for its simplicity, readability and low learning curve [32]. Supporting Python out-of-the-box can be considered an advantage in using GraphX.

```
pg = g.pageRank(tol=0.0001)

sample = pg.vertices
    .orderBy("pagerank", ascending=False)
    .limit(N)
    .collect()
```

Listing 2: Python code for finding the top N influential users using PageRank and Spark GraphX

6 Experimental Results

Table 2 summarizes our experimental results and showcases the difference between Neo4j and GraphX in terms of performance. The Neo4j implementation of PageRank in finding top 10 influential users completed in 619 seconds. On the other hand, GraphX implementation finishes in 7911 seconds, which is around 12-13 times slower. The same pattern holds true when the input size is doubled to 20 where Neo4j program requires 1342 seconds (more than 20 minutes) and Spark GraphX requires 16253 seconds (more than 4.5 hours). Last, but not least, when the input size is doubled, the computational duration increases by approximately 2 times. This tendency holds true for both Neo4j (2.17x) and GraphX (2.05x).

Table 2. PageRank Benchmark (in seconds)

N	Neo4J	GraphX
10	619	7911
20	1342	16253

7 Conclusion

This paper evaluates the performance and usability of the top 2 scalable graph processing frameworks, namely Neo4j and Apache Spark GraphX, with the help of Yelp dataset. In our tests, Neo4j outperformed Spark GraphX by a huge margin (around 12-13 times faster). However, when simplicity and usability are considered, Spark GraphX proves to be a much better choice considering that Python code is much shorter and more readable that its Cypher counterpart for writing the same query. In practice, no framework should be given preferences without considering other relevant characteristics. In fact, they could even be used in combination depending on the nature of the given problem. For example, when prototyping a solution, Spark GraphX would be favoured for the sake of code cleanliness, simplicity and Python. However, in production, a fast framework like Neo4j should be utilized and the Python prototype can be translated into Cypher code that run on Neo4j for performance gain. Last, but not least, the experimental results imply that GraphX tends to scale slightly better than Neo4j for larger input size but the current amount of data is insufficient to draw a firm conclusion. More experiments with varying input sizes will be needed to assess the scalability of these 2 frameworks, which remains a task for future research.

References

1. Bernard Marr: How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read (2018), https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/
2. Ching, A.: Giraph: Production-grade graph processing infrastructure for trillion edge graphs. ATPESC, ser. ATPESC **14** (2014)
3. Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., Muthukrishnan, S.: One trillion edges: Graph processing at Facebook-scale. Proceedings of the VLDB Endowment **8**(12), 1804–1815 (Aug 2015)
4. Corbellini, A., Godoy, D., Mateos, C., Schiaffino, S., Zunino, A.: An analysis of distributed programming models and frameworks for large-scale graph processing. IETE Journal of Research **0**(0), 1–9 (2020). https://doi.org/10.1080/03772063.2020.1754139, https://doi.org/10.1080/03772063.2020.1754139
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)

6. Dunbar, J.E., Grossman, J.W., Hattingh, J.H., Hedetniemi, S.T., McRae, A.A.: On weakly connected domination in graphs. Discrete Mathematics **167**, 261–269 (1997)
7. Elser, B., Montresor, A.: An evaluation study of bigdata frameworks for graph processing. In: 2013 IEEE International Conference on Big Data. pp. 60–67 (2013). https://doi.org/10.1109/BigData.2013.6691555
8. Facebook users worldwide 2019, Statista, `https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/`
9. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., Taylor, A.: Cypher: An evolving query language for property graphs. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1433–1445 (2018)
10. Gao, Y., Zhou, W., Han, J., Meng, D., Zhang, Z., Xu, Z.: An evaluation and analysis of graph processing frameworks on five key issues. In: Proceedings of the 12th ACM International Conference on Computing Frontiers. pp. 1–8. ACM, Ischia Italy (May 2015). https://doi.org/10.1145/2742854.2742884, `https://dl.acm.org/doi/10.1145/2742854.2742884`
11. Gleich, D.: PageRank Beyond the Web. SIAM Review **57**(3), 321–363 (Jan 2015)
12. Golden, B.L., Raghavan, S., Wasil, E.A.: The Vehicle Routing Problem: Latest Advances and New Challenges, vol. 43. Springer Science & Business Media (2008)
13. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: Graph processing in a distributed dataflow framework. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). pp. 599–613 (2014)
14. Gosling, J., Joy, B., Steele, G., Bracha, G.: The Java language specification. Addison-Wesley Professional (2000)
15. Hejlsberg, A., Wiltamuth, S.: C# language reference. Microsoft (2000)
16. Hilker, C.: Social Media für Unternehmer: Wie man Xing, Twitter, Youtube und Co. erfolgreich im Business einsetzt. Linde Verlag GmbH (2010)
17. Ian Robinson, Jim Webber, Emil Eifrem: Graph Databases 2e Neo4j. OŔeilly Media (2015)
18. Isojärvi, S.: Disco and nokia: Experiences of disco with modeling real-time system in multiprocessor environment. In: Formal Methods Europe Industial Seminar (1997)
19. Jindal, T.: Finding local experts from yelp dataset. ideals (2015)
20. Klauck, H., Nanongkai, D., Pandurangan, G., Robinson, P.: Distributed Computation of Large-scale Graph Problems. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 391–410. Proceedings, Society for Industrial and Applied Mathematics (Dec 2014)
21. Knuth, D.E.: The Stanford GraphBase: A Platform for Combinatorial Computing. ACM, New York, NY, USA (1993)
22. Krishnan, S., Gonzalez, J.L.U.: Google bigquery. In: Building Your Next Big Thing with Google Cloud Platform, pp. 235–253. Springer (2015)
23. Luca, M.: Reviews, reputation, and revenue: The case of yelp. com. Com (March 15, 2016). Harvard Business School NOM Unit Working Paper **1234**(12-016) (2016)
24. Lumsdaine, A., Gregor, D.P., Hendrickson, B., Berry, J.W.: Challenges in Parallel Graph Processing. Parallel Processing Letters **17**, 5–20 (2007)
25. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A System for Large-scale Graph Processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. SIGMOD '10, ACM, New York, NY, USA (2010)

26. Mera, D., Batko, M., Zezula, P.: Towards fast multimedia feature extraction: Hadoop or storm. In: 2014 IEEE International Symposium on Multimedia. pp. 106–109. IEEE (2014)
27. Murthy, D.: Twitter. Polity Press Cambridge, UK (2018)
28. Muth, P., Rakow, T.C.: Atomic commitment for integrated database systems. In: Proceedings. Seventh International Conference on Data Engineering. pp. 296–297. IEEE Computer Society (1991)
29. Nguyen, T.: Model the Relations between Geospatial Objects with a Graph Database. Master Thesis, Frankfurt University of Applied Sciences (2019)
30. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab, Stanford InfoLab (1999)
31. Quick, L., Wilkinson, P., Hardcastle, D.: Using pregel-like large scale graph processing frameworks for social network analysis. In: 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. pp. 457–463 (2012). https://doi.org/10.1109/ASONAM.2012.254
32. Ranking, P.: Pypi python modules ranking. URL http://pypi-ranking. info/alltime (2020)
33. Sabau, A.S.: Survey of clustering based financial fraud detection research. Informatica Economica **16**(1), 110 (2012)
34. Shirinivas, S., Vetrivel, S., Elango, N.: Applications of graph theory in computer science an overview. International journal of engineering science and technology **2**(9), 4610–4621 (2010)
35. Siek, J.G., Lee, L.Q., Andrew Lumsdaine: The Boost Graph Library: User Guide and Reference Manual. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
36. Simanowski, R.: Facebook-Gesellschaft. Matthes & Seitz Berlin Verlag (2016)
37. Tesoriero, C.: Getting started with orientDB. Packt Publishing Ltd (2013)
38. Van Rossum, G., Drake, F.L.: The python language reference manual. Network Theory Ltd. (2011)
39. Wang, J., Huang, P., Zhao, H., Zhang, Z., Zhao, B., Lee, D.L.: Billion-scale commodity embedding for e-commerce recommendation in alibaba. In: Billion-Scale Commodity Embedding for E-Commerce Recommendation in Alibaba. p. 10 (2018)
40. White, A.: JavaScript programmer's reference. John Wiley & Sons (2010)
41. Xin, R.S., Gonzalez, J.E., Crankshaw, D., Franklin, M.J., Dave, A., Stoica, I.: GraphX: Unifying Data-Parallel and Graph-Parallel Analytics. In: GraphX: Unifying Data-Parallel and Graph-Parallel Analytics (2014)
42. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: A resilient distributed graph system on spark. In: First international workshop on graph data management experiences and systems. pp. 1–6 (2013)
43. Yelp: Yelp Dataset (2019), `https://www.yelp.com/dataset`
44. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing (2012)
45. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., et al.: Apache spark: a unified engine for big data processing. Communications of the ACM **59**(11), 56–65 (2016)