HGraph: Parallel and Distributed Tool for Large-Scale Graph Processing

Wilfried Yves Hamilton Adoni*
Hassan II University of Casablanca
International University of Casablanca
Casablanca, Morocco
adoniwilfried@gmail.com

Nahhal Tarik
Faculty of Science
Hassan II University of Casablanca
Casablanca, Morocco
tarik.nahhal@univh2c.ma

Moez Krichen
Faculty of CSIT, Albaha University, KSA
ReDCAD Laboratory, University of Sfax
Sfax, Tunisia
moez.krichen@redcad.org

Abdeltif El Byed
Faculty of Science
Hassan II University of Casablanca
Casablanca, Morocco
abdeltif.elbyed@etu.univh2c.ma

2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA) | 978-1-6654-1511-8/20/$31.00 ©2021 IEEE | DOI: 10.1109/CAIDA51941.2021.9425162

***Abstract*—Graph are ubiquitous because the fields of application are varied. Well-known examples are social networks, biological networks and path-finding in road networks. Real-world graphs processing is very challenging because of 4V characteristics related to big data. They are huge to process them on single-node and the time complexity is exponential. Unfortunately, due to the lack of research, only a few systems are able to ensure the storage and quick processing of large-scale graphs. In this paper, we propose HGraph, a parallel and distributed tool which handles large-scale graphs. HGraph is build on top of Hadoop and Spark frameworks. The proposed tool provides high scalability and is adapted to easily implement algorithms for various graph problems. Experimental tests performed on real-world graphs showed that HGraph is reliable and achieves significant gain time over the state of the art of graph processing systems.**

***Index Terms*—Large-scale graph, Graph processing system, Distributed computing, Big data, Hadoop, Spark**

I. Introduction

Big data refers to the technological trend that we have seen emerge in recent years around data. This technology has aroused a great interest for industry but also the scientific community because of these techniques for storing and analyzing huge data. The analysis of complex data is a difficult task because it is strongly influenced by the structure of the data as well as the relationships between the information. An alternative is to draw inspiration from the concepts of graph theory for modeling big data, so called big graphs. Big graphs are part of NoSQL database family [1], [2]. Big graphs provide an ability to analyze a large network describing a complex phenomenon. With big graphs, we can analyze the relationships between the data. Big graphs are intuitive, scalable and flexible compared to RDBMS that use relationships between tables.

This highlights the increased use of big graphs in the modeling of several complex phenomena. The fields of application are varied: the analysis of social networks (e.g. Facebook and Twitter), the path-finding in a road network, the analysis of protein-protein interactions of biological network, etc.. Big graphs are becoming a mature abstraction covering the study of complex phenomena in various disciplines.

However, problems arise when scaling up to a large-scale graph. Indeed, the analysis of graphs composed of billions of vertices strongly connected to each other by edges is a difficult task. In some cases, the graph cannot be stored on a single machine. In this case, partitioning of the graph is an alternative. Thus, algorithms based on the ”divide-and-conquer” [3] approach are applied to partition the original graph, and then each partition is assigned to each node of the cluster. This task greatly influences the analysis phase. There are several algorithms dedicated to partitioning tasks. Some are sequential and adapted to small graphs. Others parallel and/or distributed are adapted to dense graphs. This is an NP-complete problem because until now, there is no efficient algorithm that can find an optimal solution in polynomial time [4], [5]. Moreover the choice of a partitioning algorithm is guided by the choice of the system. For systems whose computational paradigm is based on vertices (vertex-centric) [6], it is essential to opt for a partition based on the partitioning of vertices. Respectively for systems whose computing paradigm is based on edges (edge-centric) [6], it is more advantageous to use partitioning algorithms based on edge placement.

On the other hand, this task is difficult because the analysis of a large-scale graph consumes huge hardware resources. Consequently, the use of commodity machine is not appropriate for this problem. The choice of graph analysis systems is a headache because several factors must be taken into account: the size and topology of the graph, the partitioning technique, the cluster hardware resources, the programming paradigm and the algorithmic complexity of the program. Only a limited number of systems are able to perform analytical tasks on complex networks.

In this context, we proposed HGraph, a prototype tool based on the Hadoop and Spark frameworks that allows us to evolve graph analysis algorithms on a large scale. HGraph runs in-memory and switches to local disk when both physical and virtual memory are completely saturated. This allows optimal use of the cluster nodes' hardware resources. In addition, HGraph also contains utility functions for big graph analysis. The programming paradigm of the graph algorithms on HGraph is parallel and distributed and are performed of a master-slaves architecture. Thus the computation tasks are distributed on the slave nodes while the master node supervises and coordinates the workflow. This speeds up the velocity of the algorithms in the partitioning (pre-processing) and analysis (processing) phases.

The rest of this paper is structured as follows. Section 2 introduces related works about graph processing systems. In Section 3, we show the conceptual modal of our tool for large-scale graph analyzing. In Section 4 and 6, we present a case study of application in the field of formal verification and system testing. Finally, we conclude this paper.

II. Related work

Graph processing systems are classified in two categories: single-machine systems and distributed systems (running on several machines connected in a network). Pajek [7] is a case of non-distributed system for big graph analysis. Pajek has the capacity to process up to 10 million vertices. It is written in Pascal and limited to run as a Windows monolith and therefore limited to Windows operating systems which makes it difficult to extend its API to other programs. The graph exploited in Pajek is represented as a double chained list, which favors vertex insertion and deletion operations in the graph but can be slow for operations on modern CPUs with random access memory.

Other open source systems with libraries with similar functionality to Pajek's are NetworkX [8] and iGraph [9]. NetworkX provides an API for a large number of graph algorithms. It is written in Python. NetworkX provides APIs for programs written in Python and R, it offers flexibility in graph processing at the expense of performance in terms of velocity. NetworkX uses a hash table commonly known as a Python dictionary to store network vertices, edges and attributes. The use of a hash table offers flexibility in exploring the network but the run time is slower and consumes a large amount of memory. In addition, because Python programs are interpreted, their executions take much longer and consume more RAM than compiled programs.

Similar to NetworkX but different in terms of functionality, iGraph is a big graph processing system. Unlike NetworkX, iGraph is written in C and advocates performance at the expense of flexibility in data processing. The vertices and edges of the graph are stored in a Vector and indexed, which improves the speed of execution of the algorithms. On the other hand, iGraph is slow when it is a dynamic graph subject to adding or removing vertices or edges.

Compared to iGraph and NetworkX, SNAP [10], [11] optimizes big graph analysis operations. SNAP (Stanford Network Analysis Project) is a high-performance computing system for complex networks. SNAP is written in C++ but provides a module (snappy) for Python programs. SNAP implements operators optimized for the analysis of static or dynamic graphs. Similar to iGraph and NetworkX, SNAP is designed for multi-core processor-based machines with very large RAM memory. This allows to maintain the gap between performance and flexibility in the compact representation of the persistent graph in memory. Overall, SNAP is twice as fast and uses 50 times less memory than NetworkX. SNAP also uses 3 times less memory than iGraph because of the indexing of vertices and edges stored in the Vector object. On the other hand it is three times slower than iGraph.

While the non-distributed versions are designed to run on a single node requiring a large amount of RAM memory, an alternative is to perform processing in a distributed and master-slaves oriented architecture. This technique involves partitioning the original graph into several sub-graphs and distributes them to each node in the cluster. Examples of such distributed systems include PowerGraph [12], GraphX [13], Giraph [14], Pregel [15] and ExPregel [16]. Generally, the conceptual model of these systems is inspired by Hadoop's model [17]. In principle, these systems can handle a large-scale graph than a single-node system and the execution time improves proportionally to the nodes added in the cluster. But are much more difficult to implement because of the complexity related to the parallelism and distributed workflow of the cluster.

III. HGraph architecture

In this section, we present HGraph, a tool for large-scale graph analysis. HGraph is not a framework but rather a tool resulting from the combination of the main components of Hadoop [17] and Spark [18]. We first present the overall architecture of HGraph, the EPGM data model [19] allowing the modeling of structured data as graphs and the set of implemented operators.

Although there are already similar big data platforms, their configuration requires a lot of hardware resources in terms of RAM and CPU memory. For example, IBM BigInsights platform requires a minimum of 8GM of RAM, Cloudera will consume more for each added component. In this context, HGraph essentially uses the core components of Hadoop and Spark but also integrates the basic components of GraphX [13] necessary to process graph-structured data.

HGraph uses operators that optimize high performance computing by making maximum use of all the hardware components of the cluster. HGraph uses operators that optimize high-performance computing by making the most of the cluster's hardware components. In the case of in-memory computation, when the physical RAM memory reaches its saturation threshold, the computation task switches to swap memory (virtual memory). When the memory space is completely full, the computation task degrades on the local disk. HGraph thus makes it possible to gain in complexity because it optimally uses all resources of the cluster. In addition, the operators implemented in HGraph have been optimized to switch between physical memory, virtual memory and the local disk.

HGraph works in a multi-nodes cluster and is based on a master-slaves architecture, it is horizontally scalable. As shown in Figure 1, HGraph is composed of a multi-level architecture and consists of different layers: the application layer, the execution layer and the distributed storage layer. The execution of a job is carried out by the workflow that accesses the different operators implemented or reused by HGraph in the application layer. The operators available in the application layer in turn access the data structured according to the EPGM model [19] defined in the storage layer. All I/O operations are carried out in the storage layer.

A. Distributed storage layer

HGraph's storage layer is based on Hadoop HDFS [17] and Spark RDD [20]. The data is stored in a Resilient Distributed Graph (RDG) object designed specifically for storing RDD objects based on the EPGM model [19].

The RDG object thus allows a logical partitioning of the graph. Whereas, the HDFS allows distributed storage of the RDG objects on the different DataNodes of the cluster and ensures hardware failure through data replication. The EPGM model provides basic methods for graph processing . Graph processing requires fast vertex exploration and communication between vertices. The partitioning algorithms used in the storage layer are the DPHV [21]. These two methods allow to have a load balancing between cluster nodes. In addition, they propose partitions involving less network communications in the case of distributed computations based on MapReduce or Spark in-memory paradigm.

B. Distributed Workflow layer

This layer is responsible for the execution of all the tasks regardless of the computational parallelism. HGraph uses the Spark and Hadoop execution engine for the computations. When it is an iterative task, the most adapted paradigm is the in-memory computation model provided by Spark. In the case of a computation involving several I/O operations on the local disk, MapReduce model is more adapted. Cluster management is handled by Zookeeper [22] which maintains parallelism in a distributed environment. The master node partitions the job into several tasks and distributes them to all the slave nodes of the cluster.

C. Application layer

The application layer contains all the APIs for implementing an program for large-scale graph processing. The APIs available in HGraph have been implemented in Java and inherit from a set of classes available in Spark [23] and Hadoop [17]. HGraph contains two categories of operators: unary operators and binary operators. These operators allow CRUD operations on graphs. In addition, the application layer contains operators for graph partitioning (DPHV [21]) and graph analysis (MRA* and SPA* [24]) graphs.

D. HGraph operators

HGraph operators are based on the logical processing of EPGM [19] operators. These operators can return new graphs or a collection of elements. There are two types of operators: unary operators and binary operators. Unary operators are logical graph operators that take a single graph as input. Binary operators, on the other hand, perform operations on two input graphs. HGraph reuses all the logical graph operators available in Spark's GraphX [13]. The implemented operators are based on Vertex-centric [6] paradigm unlike those of GraphX which are based on the Edge-centric [6] paradigm. These new operators perform computations directly on the vertices of the graph, which facilitates communication between vertices.

Unlike Hadoop whose operators run on local disk and Spark whose operators run in memory, HGraph operators have been adapted to run first on physical memory. Then, when the physical memory is full, the computation degrades on the virtual memory. Finally, on local disk when, the RAM memory is completely saturated.

IV. Application to the coverage of large-scale automatons network with state explosion

To highlight the applicability of our tool, we have carried out performance tests of the MRA* and SPA* algorithms [24] on benchmark graph data. These data represent automatons with state explosion.

The problem of covering all final states for the formal test of systems represented by an automaton with state explosion consists in starting from an initial state (the input of the system) to find all paths that join all the final states [25], [26]. This type of problem also reveals a NP-Hard complexity [4], [5] because there is no coverage algorithm that proposes an optimal solution in polynomial time [27]. In this experiment, we apply the MRA* and SPA* algorithms implemented in HGraph to compute a parallel and distributed coverage of the final states of the automata. The tests have been run 10 times, the results presented are the mean values. The solution obtained is quasi-optimal but the computation time is significantly improved.

A. Dataset and cluster setup

The experimental tests were performed on a HGraph cluster. The cluster consists of 10 homogeneous compute nodes. Each node is equipped with a 2x Intel Xeon Gold 6130 processor (16 cores/ CPU), 192 GB RAM including 52 GB swap RAM, 700 GB SSD + 4.0 TB HDD and 10 Gbps + 100 Gbps Omni-Path Ethernet cable, running on the JVM 1.8.

Table I presents the characteristics of the graphs extracted from the data sets. This data was extracted from the set of different compliance testing models of various complex systems [1]. For each dataset, we list the size of the graph G_{size} in gigabit, the number of vertices $|V|$ associated to the states of the system, the number of edges $|E|$ modeling the transitions

[1]https://projects.info.unamur.be/vibes/mutants-equiv.html

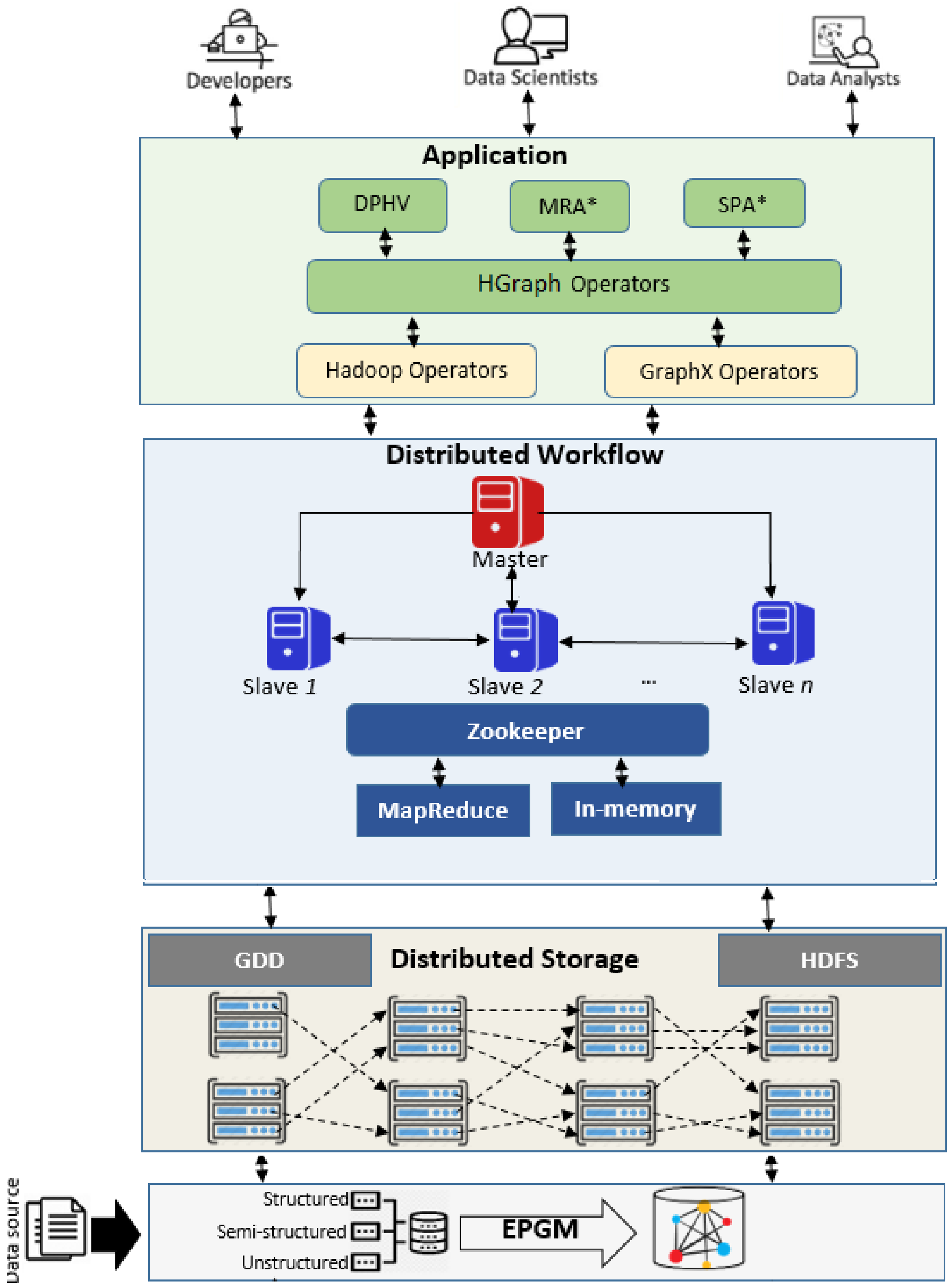

Fig. 1. Overall view of HGraph architecture

between states, the diameter of the graph D and the clustering coefficient ACC. Each automaton contains at least 5% of final states to be covered.

TABLE I
BENCHMARKS OF AUTOMATON DATA SETS

Description	|V|	|E|	**D**	**ACC**	G_{size}
ageRRN	860 000	2 360 000	56	0.4970	0.65
cptTRM	1 070 000	6 000 000	70	0.6235	1.6
elsaRR	1 508 000	9 000 000	92	0.5653	2.3

B. Comparison between sequential and parallel versions

Table II presents the results of the coverage tests on the datasets. The columns T_{A*}, T_{MRA*} and T_{SPA*} represent respectively the times taken by A* algorithm, MRA* and SPA* [24] for the coverage of the different automata. To ensure a good parallelism, the cluster has been reduced to 1 node with 16 processor cores. Thus the numbers of mappers and reducers have been fixed to 16 CPU. We can see that the parallelism improves the computation time considerably. Thus we notice that HGraph's operators are faster. For example, the coverage of the elsaRR automaton is done in 6h12min (22345 sec) using the sequential A* approach. This time is reduced to 54min (3146 sec) when the coverage is parallelized with MRA*, i.e. a gain of about 5 hours. On the other hand, MRA* performances are surpassed by SPA* which computes the coverage paths in 14min (883 sec), which represents a gain of about 1h.

TABLE II
EXECUTION TIME IN SECONDS OF A*, MRA* AND SPA*.

	Sequential	**Parallel (HGraph's operators)**	
Datasets	T_{A*}	T_{MRA*}	T_{SPA*}
ageRRN	12454	2435	362
cptTRM	17285	3146	441
elsaRR	22345	3254	883

C. Impact of CPU on runtime

Figure 2 shows the impact of the number of CPU on the computation time. It can be seen that increasing the number of CPU considerably improves the coverage time of the automata. On the other hand, the reduced phase consumes a large amount of the total time. This can be explained by the fact that during the reduce phase: (1) the concatenation of the covering paths is done iteratively for the SPA* algorithm; (2) for the MRA* algorithm, the synchronization of the shuffle operations generates a latency in the transition between the map and reduce phase. These operations imply intra-cluster communication costs because of the asynchronous exchange of data between nodes to ensure coverage in a distributed environment.

D. Impact of nodes on coverage time

Figure 3 presents the result of the run time of the MRA* and SPA* heuristics with respect to the number of nodes. In all three types of automata, we notice that the coverage times improve considerably with the increase of the number of nodes. We can also see that SPA* is faster than MRA* and uses fewer nodes to go from exponential complexity time to linear complexity time. Moreover, beyond 5 nodes the computation time of SPA* does not improve because it reaches its stationarity. On the other hand, MRA* reaches its stationarity after the addition of 7 compute nodes.

V. CONCLUSION

In this paper, we proposed HGraph a prototype tool for the implementation of graph algorithms. HGraph provides an API written in java and supports the storage of large-scale graphs one Hadoop Distributed File System and Resilient Distributed Graph of Spark. Unlike Hadoop and Spark, which perform computations on the local disk and memory respectively, HGraph runs operations on local disk and then switches to RAM and then virtual memory when memory is full. The current version of HGraph implements basic graph operators as well as DPHV algorithm and the MRA* and SPA* algorithms. HGraph is still in its embryonic phase and requires many improvements in its conceptual model. In terms of velocity, experimental results have shown that our HGraph have good performances and are also applicable to real automata.

REFERENCES

[1] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[2] A. Goel, *Neo4j cookbook harness the power of Neo4j to perform complex data analysis over the course of 75 easy-to-follow recipes*. Birmingham, England; Mumbai India: Packt Publishing, 2015.

[3] J. L. Bentley, "Multidimensional Divide-and-conquer," *Commun. ACM*, vol. 23, no. 4, pp. 214–229, 1980.

[4] V. E. Alekseev, R. Boliac, D. V. Korobitsyn, and V. V. Lozin, "NP-hard graph problems and boundary classes of graphs," *Theoretical Computer Science*, vol. 389, no. 1, pp. 219–236, 2007.

[5] K. Cameron, E. M. Eschen, C. T. Hoáng, and R. Sritharan, "The Complexity of the List Partition Problem for Graphs," *SIAM Journal on Discrete Mathematics*, vol. 21, no. 4, pp. 900–929, 2008.

[6] A. Guerrieri, "Distributed computing for large-scale graphs," Ph.D. dissertation, University of Trento, 2015.

[7] V. Batagelj and A. Mrvar, "Pajek Analysis and Visualization of Large Networks," in *Graph Drawing Software*, ser. Mathematics and Visualization. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 77–103.

[8] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," 2008.

[9] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu, "iGraph: A Framework for Comparisons of Disk-based Graph Indexing Techniques," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 449–459, 2010.

[10] J. Leskovec and R. Sosic, "SNAP: A General Purpose Network Analysis and Graph Mining Library," *arXiv:1606.07550 [physics]*, 2016.

[11] D. Hallac, C. Wong, S. Diamond, A. Sharang, R. Sosi?, S. Boyd, and J. Leskovec, "SnapVX: A Network-Based Convex Optimization Solver," *Journal of Machine Learning Research*, vol. 18, no. 4, pp. 1–5, 2017.

[12] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-parallel Computation on Natural Graphs," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 17–30.

[13] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*, ser. GRADES '13. New York, NY, USA: ACM, 2013, pp. 1–6.

[14] A. Ching, "Giraph: Large-scale graph processing infrastructure on Hadoop," in *Proceedings of the Hadoop Summit*, ser. 3, vol. 11, Santa Clara, 2011, pp. 5–9.

[15] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 135–146.

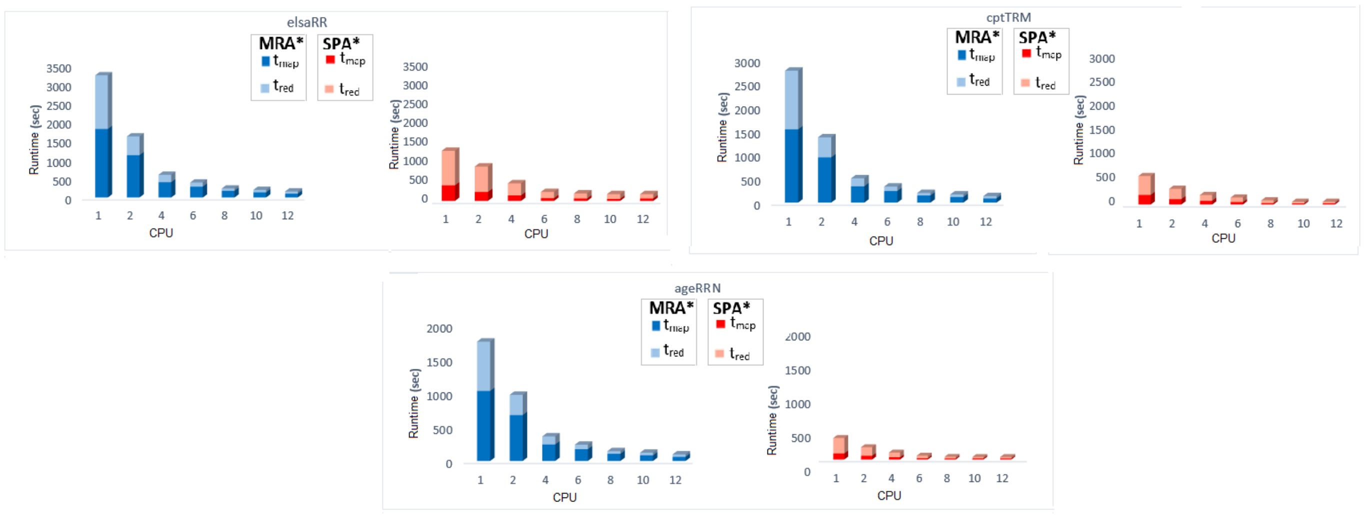

Fig. 2. MRA* and SPA* behavior with respect to the variation of the number of CPU.

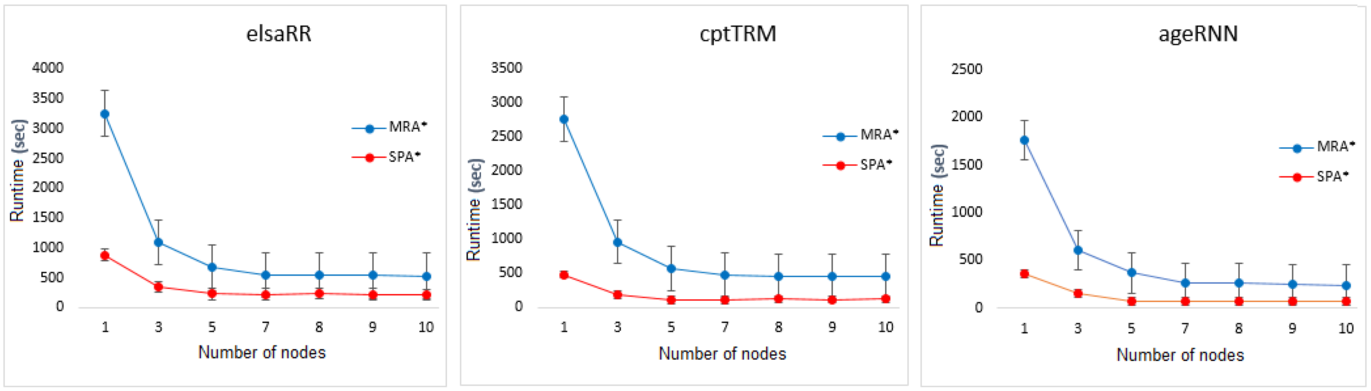

Fig. 3. Runtime versus number of nodes.

[16] M. Sagharichian, H. Naderi, and M. Haghjoo, "ExPregel: a new computational model for large-scale graph processing," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4954–4969, 2015.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE Computer Society, 2010, pp. 1–10.

[18] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, and M. J. Franklin, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[19] M. Junghanns, A. Petermann, N. Teichmann, K. G?mez, and E. Rahm, "Analyzing extended property graphs with Apache Flink," in *Proceedings of the 1st ACM SIGMOD Workshop on Network Data Analytics - NDA '16*. San Francisco, California: ACM Press, 2016, pp. 1–8.

[20] 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.

[21] W. Y. H. Adoni, T. Nahhal, M. Krichen, A. El byed, and I. Assayad, "DHPV: a distributed algorithm for large-scale graph partitioning," *Journal of Big Data*, vol. 7, no. 1, p. 76, 2020.

[22] E. Okorafor and M. K. Patrick, "Availability of Jobtracker machine in hadoop/mapreduce zookeeper coordinated clusters," *Advanced Computing*, vol. 3, no. 3, p. 19, 2012.

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[24] W. Y. H. Adoni, T. Nahhal, B. Aghezzaf, and A. Elbyed, "MRA*: Parallel and Distributed Path in Large-Scale Graph Using MapReduce-A* Based Approach," in *Ubiquitous Networking*, ser. Lecture Notes in Computer Science. Springer, Cham, May 2017, pp. 390–401.

[25] M. Krichen, "Contributions to model-based testing of dynamic and distributed real-time systems," Ph.D. dissertation, École Nationale d'Ingénieurs de Sfax (Tunisie), 2018.

[26] M. Krichen and R. Alroobaea, "A new model-based framework for testing security of iot systems in smart cities using attack trees and price timed automata," in *14th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE 2019*, 2019.

[27] M. Krichen, A. J. Maâlej, and M. Lahami, "A model-based approach to combine conformance and load tests: an ehealth case study," *International Journal of Critical Computer-Based Systems*, vol. 8, no. 3-4, pp. 282–310, 2018.